

Fuzzy-Connected 3D Image Segmentation at Interactive Speeds

László G. Nyúl,^a Alexandre X. Falcão,^b Jayaram K. Udupa^a

^a Medical Image Processing Group, Department of Radiology, University of Pennsylvania, Philadelphia, Pennsylvania 19104–6021

^b Institute of Computing, State University of Campinas, Campinas, SP 13083–970, Brazil

ABSTRACT

Image segmentation techniques using fuzzy connectedness principles have shown their effectiveness in segmenting a variety of objects in several large applications in recent years. However, one problem with these algorithms has been their excessive computational requirements. In an attempt to substantially speed them up, in the present paper, we study systematically a host of 18 algorithms under two categories — label correcting and label setting. Extensive testing of these algorithms on a variety of 3D medical images taken from large ongoing applications demonstrates that a 20–360 fold improvement over current speeds is achievable with a combination of algorithms and fast modern PCs. The reliable recognition (assisted by human operators) and the accurate, efficient, and sophisticated delineation (automatically performed by the computer) can be effectively incorporated into a single interactive process. If images having intensities with tissue specific meaning (such as CT or standardized MR images) are utilized, all parameters for the segmentation method can be fixed once for all, all intermediate data can be computed before the user interaction is needed, and the user can be provided with more information at the time of interaction.

Keywords: Image segmentation, fuzzy connectedness, 3D images, interactive processing

1. INTRODUCTION

The purpose of image segmentation is to extract object information from given images and to output this as a structure system. In many situations, the structure system consists of a single structure. Segmentation is needed directly or indirectly for most of the operations done on images. It is also the most difficult of all image operations.

Segmentation may be thought of as consisting of two related tasks — *recognition* (or detection) and *delineation*. Recognition is the high-level task of determining roughly the whereabouts of the object in the given image. Delineation is the low-level task of determining the precise spatial extent of the object and its point-by-point graded composition. In most recognition tasks, trained human operators outperform any computer algorithms. On the contrary, computer algorithms exist for delineation that are more precise, accurate, and efficient than human delineation of object regions or boundaries. Human delineation that specifies graded objectness is impossible at present. Recognition and delineation are not completely disparate steps in segmentation.

Approaches to recognition may be broadly classified into two groups: *automatic* and *human-assisted*. In automatic recognition methods (either knowledge-based^{1,2} or atlas-based^{3,4}), a question arises as to what to do in case of failures. Completely automatic methods that are fool-proof and that have been demonstrated to work correctly (or with acceptable errors) routinely in trials involving a large number of imaging studies do not seem to have been constructed yet. The premise in human assisted methods then is that, often a simple help from an operator (on a per-study basis) is sufficient as a recognition aid. Therefore, if we can make this process efficient, then the uncertainties of the automatic methods can be overcome, and we have a solution that is practical. This help may be in the form of (i) specification of a few “seed” points in the object region or on its boundary, (ii) indication of a box enclosing the object, or (iii) clicking of a mouse button to accept a real object that has been delineated or to reject a false object.

Image segmentation research for delineation spans nearly four decades.⁵ Usually delineation itself is considered to be the total segmentation problem. That is, whatever is output by the delineation method is considered to represent the object of interest. Two classes of methods exist for delineation: *boundary-based*^{6–10} in which the output structure represents the boundary of the object, and *region-based*^{11–14} in which the output structure represents the region occupied by the object. In both groups, the output structure may be hard (crisp) or fuzzy. In the hard case, the object grade is ignored and a binary output as a (hard) set of boundary or space elements is sought. In the fuzzy case, object heterogeneity is taken into account, and often the grade of the object is retained in the output, providing a fuzzy boundary or region output. Any of the two strategies

of recognition — automatic and human-assisted — can be combined with any of the four groups of delineation approaches, resulting in eight classes of possible segmentation strategies.

Images are by nature fuzzy. Approaches to object information extraction from images should therefore attempt to retain uncertainties as realistically as possible. Local fuzziness has been accounted for in fuzzy delineation approaches in the past. However, the notion of “hanging togetherness” of image elements in a global fuzzy sense for object definition has been lacking. In an attempt to capture this notion, we developed a theoretical and algorithmic framework called fuzzy connectedness.¹⁵

Segmentation techniques using fuzzy connectedness principles¹⁵ have been used successfully in several large applications (involving over 1500 3D studies) in recent years: MR brain image analysis for quantifying white matter lesions,¹⁶ segmentation of vessel structures and artery-vein separation in MRA,^{17,18} craniofacial muscle segmentation,¹⁹ and mammographic fibroglandular density quantification.²⁰ However, one problem with the fuzzy connected object segmentation algorithms has been their excessive computational requirements. This stems from the need to determine the “strength of connectedness” of every possible path between every possible pair of image elements in the image domain.¹⁵ In the brain lesion segmentation application, for example, the segmentation method¹⁶ took approximately one hour per study (on a Sun Sparc 20), and most of this time was spent on segmenting fuzzy connected objects. In order to make this robust and effective technique practically more useful, we investigated its bottleneck operations and found several ways for substantial improvements, to the point that the entire segmentation process can be carried out at close to interactive speeds on modern PCs.

A preliminary effort in this direction was made in our group recently²¹ using Dijkstra’s algorithm in place of the original dynamic programming algorithm suggested in.¹⁵ Based on 2D phantoms and two 2D MR slice images, this work²¹ demonstrated that a 7–8 fold speed-up in fuzzy connectedness object extraction can be achieved. Since the segmentation tasks considered in this paper were somewhat artificial and not in 3D, it was not clear how the observed speed up will hold for segmenting real 3D objects in routine practice. Thus, motivated by this result, in the present paper, we study systematically a host of optimal graph search algorithms on a variety of 3D medical images taken from several large ongoing applications and demonstrate that a 20–360 fold speed up is achievable with a combination of algorithms and fast modern PCs in actual routine practice.

The principles of fuzzy connectedness are described briefly in Section 2. In Section 3, we present the algorithms and the data structures studied and used in achieving the improved speed. The materials utilized, the testing environment, and the results are detailed in Section 4. Finally, we state our concluding remarks in Section 5.

2. FUZZY CONNECTEDNESS PRINCIPLES

We refer to an n -dimensional digital image as a *scene* and represent it by a pair $C = (C, \mathbf{f})$, where C is a rectangular n -dimensional array of spatial elements (*spels* for short) and \mathbf{f} is a function (called *scene intensity*) which assigns to every spel a vector of m integer intensity values. The spels are pixels for $n = 2$ and voxels for $n = 3$. We use the notation $c = (c_1, c_2, \dots, c_n)$ to denote the coordinates of any spel $c \in C$ in the array.

2.1. Fuzzy Adjacency and Affinity

Independent of any image data, we think of the digital grid system defined by the spels as having a *fuzzy adjacency relation*. This relation assigns to every pair (c, d) of spels a value between zero and one. The closer c and d are spatially to each other, the greater is this number. This is intended to be a “local” phenomenon. How “local” it ought to be should perhaps depend on the blurring property of the imaging device. We denote the fuzzy adjacency relation by α and the degree of adjacency assigned to any spels (c, d) by $\mu_\alpha(c, d)$.

Now consider the grid points (spels) as having scene intensities assigned to them. That is, we are given a scene $C = (C, \mathbf{f})$. We define another local fuzzy relation called *affinity* on spels denoted by κ . The strength of this relation between any spels c and d , denoted $\mu_\kappa(c, d)$, lies between zero and one, and indicates how the spels “hang together” locally in the scene. $\mu_\kappa(c, d)$ is determined based on $\mu_\alpha(c, d)$, as well as on how similar the intensities or intensity-based properties at c and d are. The functional form of affinities plays an important role in algorithmic efficiency as discussed later on. The properties of fuzzy affinity relations are studied extensively and guidance as to how to setup fuzzy affinities in practical applications is given in.²² The functional forms actually utilized in the present paper are given in Section 4.1 Eqs. (9)–(14).

2.2. Fuzzy Connectedness and Fuzzy Objects

Our aim is to capture the global phenomenon of “hanging togetherness” in a global fuzzy relation on spels called *fuzzy connectedness*, denoted K . The strength of this relation $\mu_K(c, d)$ between any spels c and d , indicating the strength of their connectedness, lies between zero and one, and is determined as follows: There are numerous possible “paths” within the scene domain C between c and d . Each path for our purposes is a sequence of spels, starting from c and ending in d , with the successive spels being nearby. (In general, “nearby” means that the successive spels v and v' are such that $\mu_\alpha(v, v') \neq 0$.) We think of each pair of successive spels as constituting a link and the whole path to be a chain of links. We assign a strength (between zero and one) to every path which is simply the smallest pairwise spel affinity along the path. This indicates the weakest link in the chain. Finally, the strength of connectedness between c and d is the strength associated with the strongest of all paths between c and d .

A *fuzzy connected object* O of C of strength $\theta_x = [x, 1]$, $0 \leq x \leq 1$, and containing a spel o consists of a pool $O \subset C$ of spels together with a value indicating “objectness” assigned to every spel. O is such that $o \in O$, for any spel c and d in O , the strength of connectedness between them $\mu_K(c, d) \geq x$, and for any spels $c \in O$ and $e \notin O$, the strength $\mu_K(c, e) < x$. The value of objectness assigned to spels in O varies between zero and one and the value assigned to spels outside O is zero. See¹⁵ for a precise mathematical definition. There are several choices as to how to assign objectness to spels in O , see.^{15,22} A *K-connectivity scene* of a scene $C = (C, \mathbf{f})$ with respect to a spel $o \in C$ and fuzzy affinity κ in C is a scene $C_o = (C_o, \mathbf{f}_o)$ such that $C_o = C$, and for any spel $c \in C$, $\mathbf{f}_o(c) = \mu_K(o, c)$.

3. ALGORITHMS

To explain our motivation for speeding up the fuzzy connectedness algorithm and for systematically evaluating the new algorithms, an application context becomes necessary. We shall utilize our application relating to the segmentation and quantification of multiple sclerosis (MS) lesions of the brain¹⁶ from dual-echo T2 and proton density (PD) scenes for these purposes. The operator recognition help required in this application is provided in two sessions: (i) by specifying a few seed points for indicating the white matter (WM), gray matter (GM), and cerebrospinal fluid (CSF) (and not for lesions), and (ii) by accepting/rejecting each potential fuzzy 3D lesion object detected. After the first session of specifying the seed points, various 3D fuzzy connected objects representing the WM, GM, CSF and the lesions are delineated automatically within the two-component (T2, PD) scene. Upto nine 3D tracking of fuzzy objects is called for in this method (see¹⁶ for details) including the lesion objects delineated at different strengths. Consequently, in a clinical trial set up, when experimenting with the different parameters of the T2-PD protocol, repeated segmentation experiments involving a couple of patient data sets can take up from several hours upto several days between the first and the second interactive session. A speed up of the basic fuzzy connectedness algorithm can substantially facilitate practical applications such as MS lesion analysis in clinical trials.

Generally, in the fuzzy connectedness segmentation method, there are two major computational tasks: (i) computing the fuzzy affinity relations, and (ii) computing the fuzzy connectedness relations. In the original fuzzy connectedness algorithms,^{15,16} (i) and (ii) were not separate tasks. In this paper, we will refer to (i) as “affinity computation” and (ii) as “tracking” a fuzzy object. Both tasks can be optimized and a dramatic improvement can be achieved in their speed. Although affinity computation is straightforward, it has to be done separately for each object since the intensity features used for different objects may be different. The $[0, 1]$ interval is “scaled up” to a certain integer range (say, $[0, 8191]$) and the real-valued functions (such as Gaussian) are pre-computed and lookup tables are used to avoid the use of floating point arithmetic. This step was already optimized for speed. However, if we know certain properties of the scenes and the objects in advance, we can minimize affinity computation by avoiding any computations for spels that will definitely not be used during tracking. We will discuss several strategies along these lines. Task (ii), the computation of fuzzy connectedness values, is accomplished via a graph search algorithm, which may be speeded up if efficient search strategies and data structures are used in the implementation. Most of the results presented here address this efficiency problem. In the rest of this section, we shall first present two groups of algorithms (in Section 3.1) and then describe the strategies for speeding up affinity (in Section 3.2) and fuzzy connectedness computation (in Section 3.3).

3.1. Algorithms

Computing the fuzzy connectedness values for a fuzzy object is a variation of the single-source-shortest-path problem. This important graph theory problem and the associated algorithms are described elsewhere.²³ We now present two groups of algorithms in a general form so that the different variants, strategies and data structures used, all can be described in the same framework. These are modified versions of well-known algorithms for solving the single-source-shortest-path problem. The first group corresponds to the *label-correcting* algorithms²³ while the second to the *label-setting* algorithms.²³ Both

approaches are iterative. They assign tentative strength labels to spels at each step; the strength labels are estimates of (i.e., lower bounds on) the strengths of the strongest paths from the source (seed spels) to the individual spels. The approaches vary in how they update the strength labels from step to step and how they “converge” toward the strengths of the strongest paths. Label-setting algorithms designate one label as permanent (optimal) at each iteration. In contrast, label-correcting algorithms consider all labels as temporary until the final step, when they all become permanent. The label-correcting algorithms are more general and apply to all classes of problems. The label-setting algorithms are much more efficient, but are applicable only to special situations. Fortunately, both approaches (and so, the more efficient label-setting algorithms) are applicable to the fuzzy connectedness problem.

The original algorithms described in¹⁵ fall in the label-correcting class (Algorithm 1 below). This algorithm is an application of dynamic programming. The algorithms of the label-setting group (Algorithm 2 below) are basically various implementations of Dijkstra’s well-known algorithm. Both algorithms take as input C , o , and κ as defined in Section 2, and they output a K -connectivity scene $C_o = (C_o, f_o)$ of C . An nD array representing C_o and a queue Q of spels are used as auxiliary data structures. We refer to the array itself by C_o for the purpose of the algorithms.

Algorithm 1. Label-Correcting

```

begin
1.  set all elements of  $C_o$  to 0 except  $o$  which is set to 1;
2.  push all spels  $c \in C_o$  such that  $\mu_\kappa(o, c) > 0$  to  $Q$ ;
3.  while  $Q$  is not empty do
4.    remove a spel  $c$  from  $Q$ ;
5.    find  $f_{\max} = \max_{d \in C_o} \{\min(f_o(d), \mu_\kappa(c, d))\}$ ;
6.    if  $f_{\max} > f_o(c)$  then
7.      set  $f_o(c) = f_{\max}$ ;
8.      push all spels  $e$  such that  $f_{\max}$ ,  $f_o(e)$ , and  $\mu_\kappa(c, e)$ 
        satisfy certain conditions to  $Q$  (see Section 3.3.1);
9.    endif;
10. endwhile;
end

```

Algorithm 1. Label-Correcting

```

begin
1.  set all elements of  $C_o$  to 0 except  $o$  which is set to 1;
2.  push  $o$  to  $Q$ ;
3.  while  $Q$  is not empty do
4.    remove a spel  $c$  from  $Q$  for which  $f_o(c)$  is maximal;
5.    for each spel  $e$  such that  $\mu_\kappa(c, e) > 0$  do
6.      set  $f_{\min} = \min\{f_o(c), \mu_\kappa(c, e)\}$ ;
7.      if  $f_{\min} > f_o(e)$  then
8.        set  $f_o(e) = f_{\min}$ ;
9.        if  $e$  is already in  $Q$  then
10.         update  $e$  in  $Q$ ;
11.       else
12.         push  $e$  to  $Q$ ;
13.       endif;
14.     endfor;
15.   endwhile;
end

```

3.2. Affinity Computation

We tested two strategies for computing the fuzzy affinity relations: (S1) pre-compute affinities for all pairs of spels having non-zero adjacency (i.e., neighboring spels in our implementation) before tracking commences, and use as lookup tables during tracking, and (S2) compute affinities on-the-fly, when it is needed by the tracking algorithm for a certain pair of spels (and store the computed values so that they can be reused by simple table lookup if referenced again). (S1) is a good choice if the threshold for the fuzzy object is not known in advance and we want to experiment with different thresholds to find the optimum for the given fuzzy object. It may also be used if the connectedness values are to be computed for the whole scene (e.g., to be thresholded at different thresholds to extract fuzzy objects of different strengths, as used in our lesion quantification method). However, if a threshold is known in advance (e.g., the final threshold for the object, or the smallest from several thresholds to be used in case of the computation of fuzzy objects of different strengths), computing the affinity values for pairs of spels that are not used in tracking is a waste of time, and so (S2) may be more efficient. (S2) may also be more efficient when we can restrict the computation of the affinity and connectedness relations to the foreground of the image only (which, in our example, is about 25% of the total scene domain). The details of affinity computation are not given here. They directly follow from the functional forms of the affinity relations. The functional forms utilized in our examples are given in Section 4.1 Eqs. (9)–(14).

3.3. Tracking

Here we have many more choices than we had for affinity computation. Although there is no single “best” strategy (different strategies may be the “best” for different kinds of applications, and different affinity relations), we found a few variants that

are good approximations to the (possibly) best in most cases. We now present the details of the different strategies and data structures used for tracking. We labeled the methods according to a four-digit numbering scheme wherein each digit represents some property of the method/data structure used. These labels will be used throughout Section 4.

The different data structures we have utilized include: queue, priority queue, various heaps, LIFO and FIFO lists, hash tables with different hashing functions and table sizes. The strategies used are: different criteria for putting a spel in a queue, using different data structures to maintain the priority queue, using different hash functions (i.e., hashing by geometry-based properties or by affinity-based properties), and strategies for addressing the memory. A discussion of these data structures is beyond the scope of this paper. See²³ or any standard textbook on data structures for a detailed discussion. Also, we describe the algorithms without proving that they actually terminate and that their output is what is expected. Some of these proofs can be found in the literature (e.g., that the dynamic programming algorithm terminates), while others are easy to prove.

3.3.1. Label-Correcting Algorithms

A queue is a special kind of ordered list (set) in which elements are inserted at one end and deleted from the other end. It means that the elements stored in the queue are examined in a first-in, first-out (FIFO) order. The condition used in Step 8 of Algorithm 1 in the original fuzzy connectedness algorithm¹⁵ was

$$\mu_k(c, e) > 0. \quad (1)$$

This means that whenever the label of a spel c is updated, all neighboring spels e (with non zero affinity) are pushed to the queue Q , because the strongest-so-far path to e might be improved through c . This resulted in many unnecessary push operations (and therefore unnecessary pop operations later), since, in most cases, a change in a spel label has effect only on a few neighbors.

The implementation used in¹⁶ achieved a big practical improvement by replacing the condition in Step 8 of Algorithm 1 by

$$f_{\max} > f_o(e). \quad (2)$$

Carefully examining all possible relationships between f_{\max} , $f_o(e)$ and $\mu_k(c, e)$, we established an optimal condition for Step 8 of Algorithm 1:

$$f_{\max} > f_o(e) \quad \text{and} \quad \mu_k(c, e) \geq f_o(e). \quad (3)$$

This is a theoretically optimal condition. Since it means evaluating a more complicated expression, in real applications, its use may not really pay off. Especially, if additional techniques are involved to improve the tracking speed, such as an additional bit-array to keep track of whether or not a spel e is already in the Q . If e is already in the Q , it need not be duplicated. This latter addition can be used with any of the above three conditions in Eqs. (1)–(3). Our labels are $00b0$, $00b1$, and $00b2$ for the algorithms using condition in (1), (2), and (3), respectively, where b is either 0 (no additional bit-array) or 1 (use additional bit-array). The following sections describe the different data structures and strategies used for maintaining the priority queue in the more efficient label-setting algorithms (Algorithm 2).

3.3.2. Label-Setting Algorithms Using Binary Heap

A heap is a data structure for storing a collection of elements when each element has an associated key. For every element c other than the root, the heap satisfies the following property:

$$\text{key}(\text{parent}(c)) \geq \text{key}(c). \quad (4)$$

The main operations on the heap we use in this paper are: (i) *insert* – insert an element in the heap, (ii) *find-max* – find an element with the maximum key in the heap, (iii) *delete-max* – delete the element with the maximum key from the heap, (iv) *increase-key* – increase the key of an element already in the heap.

The binary heap always satisfies the following property that is maintained by each operation on the heap. Each node in the binary heap has at most 2 child nodes, which are assumed to be ordered from left to right. New nodes are added to the heap in an increasing order of depth values, and for the same depth value nodes are added from left to right. This means that the binary heap can be viewed as a complete binary tree. Although, the heap terminology uses the usual tree terms (root, parent, child node), and in many cases the heap is indeed implemented as a tree, its contiguous structure allows it to be implemented as an array with quite efficient operations. Finding elements other than the one with the maximum key (i.e., the root) in a heap is usually not really efficient unless some other helper structures are used for the search. Our versions of the tracking algorithm using heaps use different helper structures and have different efficiencies.

We represent the priority queue Q in Algorithm 2 by a binary heap. The key of a spel c in Q is the tentative strength label $f_o(c)$ of c at the time it is inserted into Q . Since the root stores the element with the largest key, Step 4 is the combination of a *find-max* and a *delete-max* operation. The push operation in Steps 2 and 12 is the *insert* heap operation, and the update in Step 10 is the *increase-key* operation. In the first version of binary heap (denoted by 1000), we do not keep track of whether an element is already in the heap, and we do not perform search in the heap for an already stored element, so we always insert a new instance of e in Steps 10 and 12, even if it means duplication. In another version (denoted by 1010) we maintain an additional pointer array, which, for every voxel v stores the position of v in the heap (or a marker that v is not in the heap). This array is used in the test in Step 9 and by the update (*increase-key*) operation in Step 10. We will come back to the memory versus speed issue later.

The tracking algorithms in our final group using binary heaps are more memory-conserving. They use hash tables of various sizes with various hash functions to keep track of the positions of the voxels in the heap to make the *increase-key* operation used in the update (Step 10) efficient. However, maintaining these hash tables needs additional computation, and different hash functions work differently depending on the distribution of the key values assigned to the input data (i.e., depending on the affinity values and on the geometric structure of the fuzzy object tracked). The four hash functions we implemented assign a key value to a heap element c (a spel, in our case) using the following formulas:

$$\text{key}(c) = ((c_3 * \text{height} + c_2) * \text{width} + c_1) \text{ modulo TS}, \quad (5)$$

$$\text{key}(c) = (c_3 + c_2 + c_1) \text{ modulo TS}, \quad (6)$$

$$\text{key}(c) = (c_3 * c_2 * c_1) \text{ modulo TS}, \quad (7)$$

$$\text{key}(c) = (c_3 \text{ XOR } c_2 \text{ XOR } c_1) \text{ modulo TS}, \quad (8)$$

where c_1, c_2, c_3 are the coordinates of the spel (voxel) c , height and width are the dimensions of a slice of the scene, TS is the size of the hash table, and XOR is the bit-wise exclusive or operation. The first hash function is the commonly used linear addressing scheme of elements in multidimensional arrays. The remaining represent some simple ways of combining the three coordinates into a single key value. TS depends on the actual hash function and was set up by studying the distribution of the key values with different values of table size. For Eqs. (6) and (8), we used $\text{TS} = 768$ and $\text{TS} = 256$, respectively, that creates a separate hash bin for each possible combination of 8-bit coordinates (i.e., the modulo operator never merges two distinct values). For Eqs. (5) and (7), the range of possible values is quite large and hashing is really necessary. We selected $\text{TS} = 8191$ for these cases, since this is reasonably small yet it results in a fairly uniform distribution of hash keys in our application (hence efficient hashing). We use the labels 1020, 1021, 1022, and 1023, respectively, for the algorithms corresponding to Eqs. (5)–(8).

3.3.3. Label-Setting Algorithms Using Fibonacci Heap

The Fibonacci heap is a data structure that allows the heap operations to be performed more efficiently than binary heaps. The *insert* and *increase-key* operations are very efficient on Fibonacci heaps because the “real work” to re-arrange the tree is delayed until the next *delete-min* operation. This makes Fibonacci heaps more efficient when the number of *insert* and *increase-key* operations is sufficiently large compared to the number of *delete-min* operations. We used the same six versions of the tracking algorithm as with binary heaps, and the corresponding labels differ only in the second digit: 1100, 1110, 1120, 1121, 1122, 1123. With both types of heap structures, when hash tables are used to help the search for arbitrary elements in the heap, the hash function assigns the key value based on the coordinates of the spel, which we may call hashing based on geometry. In the next section, we describe another algorithm, which uses hashing based on affinity properties.

3.3.4. Label-Setting Algorithms Using Dial’s Implementation

Dial²⁴ gave a practically very efficient implementation of Dijkstra’s shortest path algorithm. We modified his data structure and algorithm to compute fuzzy connectedness values. In our problem, since each voxel’s label (connectedness value) is bounded by the maximum possible affinity value (which in our cases is at the most 65535), we can use the temporary connectedness strength label directly as a hashing key. We maintain a list of buckets, numbered $0, 1, \dots, A$, where A is the maximum possible (scaled integer) affinity value in the application, and each bucket k stores all nodes with the temporary connectedness value equal to k . The nodes within the same bucket are stored in a doubly linked list, so that both LIFO and FIFO type of operations can be performed on it. When a spel c is entered into the priority queue Q (*insert*), its actual temporary label $f_o(c)$ is used as a hashing key, and c is put into the end of the corresponding bucket’s list. When a maximum-keyed spel is to be removed from Q (*find-max*, *delete-max*), the first (FIFO) or the last (LIFO) element is removed from the largest keyed non-empty bucket. Due to the properties of the doubly linked list, all these operations can be carried out quite inexpensively. If a spel needs to be updated (*increase-key*), first it needs to be found in the bucket it is currently stored in, then be removed from the bucket and inserted

into the new bucket (corresponding to the increased key). For this step, a search has to be made through the bucket's list, which is the most expensive part, then removing and re-inserting are simple. We tried two versions of the above described algorithm: with the LIFO version of *find-max* (denoted by 1200), and with the FIFO version (denoted by 1201). We tested also a version, where an additional pointer array is used (as with the heaps) to make direct access of elements in the queue possible without searching through the bucket lists. This latter version is referred to by the label 1210.

4. EXPERIMENTS, RESULTS, AND DISCUSSION

4.1. Image Data and Task

For the results presented in this paper, we utilized MR image data acquired with the following fast spin-echo, dual echo protocol: GE 1.5T Signa Scanner, TR = 2500 ms, TE = 18 ms/90 ms, field of view = 22 cm, matrix size = 256×192 , slice thickness (contiguous) = 3 mm, and pixel size = $0.86 \text{ mm} \times 0.86 \text{ mm}$. Ten scenes, with two intensity values per voxel, acquired as per the above protocol of ten different persons of no known brain disorders, were utilized in our evaluation of speed. The task was to classify the component tissues (white matter (WM), gray matter (GM), and cerebrospinal fluid (CSF)) of the brain. This is a common step required in many neurological applications. All data sets have been first standardized by our MR intensity standardization method²⁵ to make them have well defined intensity scale for the main foreground objects (i.e., the brain tissues, in our case). The parameters for the affinity relations for the different fuzzy objects (tissues) were estimated from a separate set of pre-segmented datasets of the same protocol that underwent the same intensity standardization. The segmentation of these training sets was verified and manually corrected by a radiologist when found necessary.

The actual number of slices for the 10 MRI scenes varied between 47 and 55, and the average number of voxels per scene was 3,342,336. The average segmented volume in voxels (\pm standard deviation) was 217,381 ($\pm 26,657$) for WM, 260,574 ($\pm 23,398$) for GM, and 167,171 ($\pm 31,666$) for CSF, that is 6.50% ($\pm 0.71\%$), 7.81% ($\pm 0.77\%$), and 4.99% ($\pm 0.82\%$), respectively, of the scene domain. Among the three objects, WM is the simplest in shape, GM shares a complex surface with the peripheral CSF, and CSF also shares another similar surface with the dura (considered as background in this application). The peripheral CSF has very fine structures, and the partial volume effects are most severe on the CSF-background and CSF-GM boundaries. Due to the differences in these structural shapes, the actual tracking time is not proportional to the volume of the fuzzy objects.

The following functional forms were used for the fuzzy affinity relations in all our experiments. We consider the given vector-valued scene in the MRI case to be equivalent to two scenes $C_{PD} = (C, f_{PD})$ and $C_{T2} = (C, f_{T2})$.

$$f_{\text{ave}}(a, b) = \frac{1}{2}(a + b), \quad (9)$$

$$f_{\text{reldiff}}(a, b) = \begin{cases} \frac{|a-b|}{a+b}, & \text{if } a + b \neq 0, \\ 0, & \text{otherwise,} \end{cases} \quad (10)$$

$$G(x, m, \sigma) = \frac{1}{(2\pi)^{\frac{1}{2}} \sigma} e^{-\frac{1}{2\sigma^2}(x-m)^2}, \quad (11)$$

$$g_{pq}(c, d) = G(f_q(f_p(c), f_p(d)), m_{pq}, \sigma_{pq}), \quad (12)$$

$$g_p(c, d) = \min_q g_{pq}(c, d), \quad (13)$$

$$\mu_\alpha(c, d) = \mu_\alpha(c, d) ((g_{PD}(c, d))^{w_1} (g_{T2}(c, d))^{w_2}). \quad (14)$$

In these expressions, G is a Gaussian function with mean m and standard deviation σ , p and q are indexes such that $p \in \{PD, T2\}$, and $q \in \{\text{ave}, \text{reldiff}\}$, and w_1 and w_2 are nonnegative weights such that $w_1 + w_2 = 1$. In our example application, we use $w_1 = 2/3$ and $w_2 = 1/3$ for WM and GM objects, and $w_1 = 1/3$ and $w_2 = 2/3$ for CSF. For computational simplicity, we used the six-adjacency relation for α , weighted by the appropriate voxel dimensions. That is,

$$\mu_\alpha(c, d) = \begin{cases} 1, & \text{if } c = d, \\ \frac{\min(s, \|c-d\|)}{\|c-d\|}, & \text{if } c \text{ and } d \text{ are 6-adjacent,} \\ 0, & \text{otherwise,} \end{cases} \quad (15)$$

where s represents the length of a side of the square pixels and $\|c - d\|$ represent Euclidean distance between the centers of c and d .

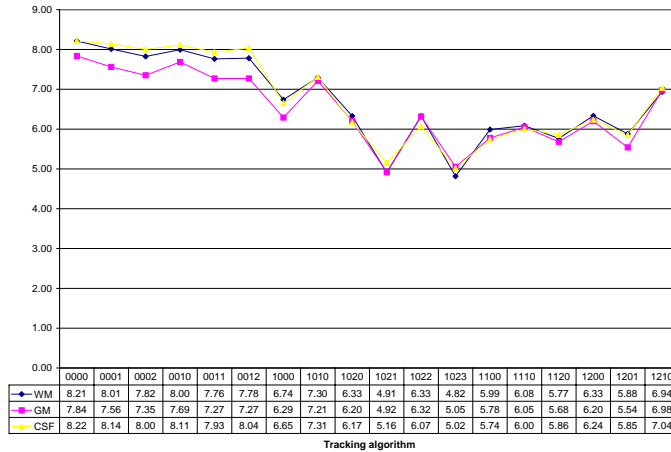


Figure 1. The ratios of the run time on the SUN to that on the PC, averaged over 3 input MRI scenes, for the various algorithms. The actual values of the ratios for the different objects are also listed at the bottom.

4.2. Comparisons

4.2.1. Hardware

For several years, SUN workstations have been used in our MS lesion quantification projects to perform fuzzy object segmentation. Although, mostly the processing is still done on these workstations, we are slowly moving to the faster PC platform. The previously reported results were according to the SUN machine’s performance, so we made a speed comparison between the two hardware configurations by running the same programs with the same parameters on the same data sets on the two machines. The first configuration is a SUN SPARCstation-20 with four 50 MHz CPUs, 256 MB RAM, running under the Solaris 2.5 operating system. The second configuration is a Gateway2000 PC with one 300 MHz PentiumII CPU, 256 MB RAM, running under the Linux 2.0 operating system. Although the SUN machine had multiple processors, the fuzzy tracking program has not made any use of this feature. The program was run in quasi-single-user mode, which means that no other processes were using the CPU and RAM that would force the operating system to use virtual memory on hard disk (swapping). Additionally, instead of the actual running time, the CPU time of the process was measured, so the multiprocessing environment had no effect on the results. The same source code was compiled with the same GNU C compiler on both machines with the same optimization options. The file I/O operations were not included in the timing measurements. By setting up this environment, we assured that the difference between the measurements is really due to the hardware performance. The ratios of the run time on the SUN to that on the PC are shown in Figure 1, averaged over 3 MRI (brain) input scenes, for the various algorithms. Our main observation is that, on average, the PC is approximately 7 times faster than the SUN workstation, when using the fastest algorithms (1210, discussed later), about 6 times faster when using close-to-best algorithms (1110 and 1200), and 7 to 8 times faster with the slower label-correcting algorithms.

Since our fuzzy tracking program is written in a way that it does not use floating point operations (except when creating a few integer lookup tables which takes place outside the tracking loop), and it does not use any kind of special features (such as graphics accelerators, multiple processors), the 6:1 speed ratio may be explained simply by the difference in the CPU clock frequencies. But, it does not mean that the PC should be 6 times faster for all kinds of applications. It is well illustrated here that the speed factor between the two hardware platforms is more when using the label-correcting algorithms (8:1) that use about 20–30 times more iterations than the label-setting algorithms (6:1).

For the rest of this paper, all results shown were measured on the PC platform.

4.2.2. Software — Affinity Computation

In this comparison, since our aim was to measure the speed improvement that can be gained if the affinity computation can be restricted in some way, we used our fastest tracking algorithm (1210) to do the actual tracking in each case.

The results in Table 1 show that when affinity is computed on-the-fly, the tracking takes about half the time as compared to the step-by-step method. Since the affinity values for the background spels are not computed, it would save about 75% of computation. However, when processing on-the-fly, additional function calls for each referred affinity value create some

Table 1. Running time for fuzzy object tracking including affinity computation. Comparison of the cases of pre-computed affinities and on-the-fly computation for several objects. The reported times are in seconds.

how	what	WM	GM	CSF
step-by-step	Aff + Con	71.44	72.32	77.86
	Aff only	67.01	65.95	69.73
on-the-fly	Con(+Aff) whole	35.73	46.81	58.87
	Con(+Aff) fg. only	35.31	46.38	52.68
	Con(+Aff) with thr.	11.71	14.87	7.78

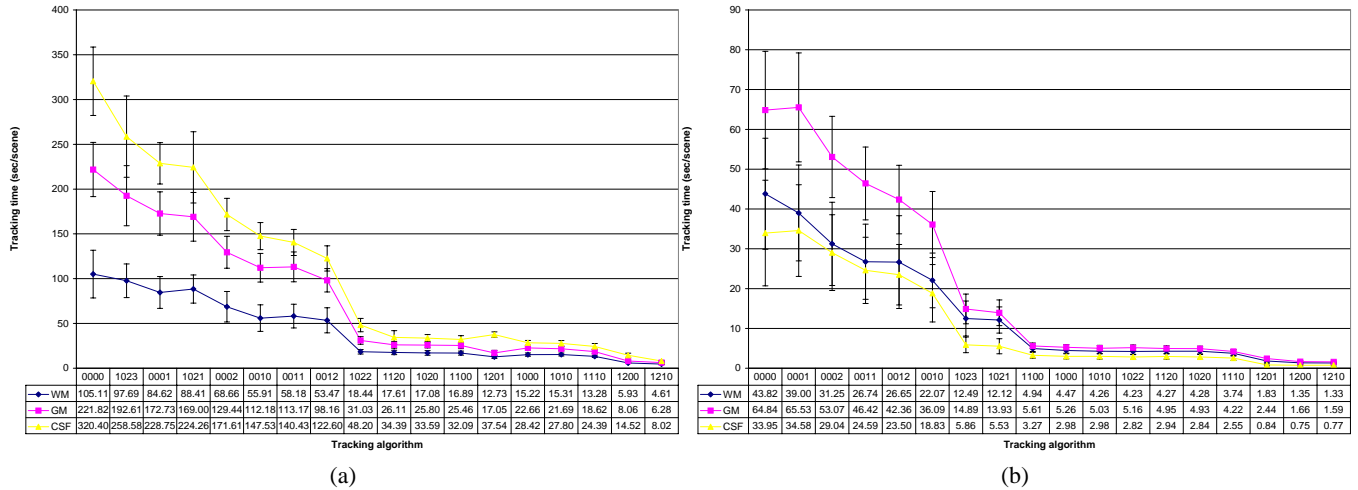


Figure 2. Average tracking time (in seconds) over 10 MRI scenes for different algorithms for different fuzzy objects using pre-computed affinities without (a) and with (b) pre-determined threshold for the connectivity scenes. The actual times for each algorithm for each object are also listed at the bottom. The vertical bars indicate a one standard deviation upper and lower limits from the mean tracking time. Note the difference in the ranges on the time axes between (a) and (b).

overhead, hence the real saving is less. Further, the on-the-fly algorithms can make essential use of pre-determined thresholds, and the entire computation may take less than 33% for WM and GM, and 15% for CSF, as compared to that without thresholds (less than 20% for WM and GM, and 10% for CSF, as compared to the step-by-step method). Since the “on-the-fly” method computes affinity (and connectedness) values for only those spels that are actually involved in tracking, the computation is already confined to the foreground region of the image. There is only a slight change in the timing when the program is forced not to compute anything outside the foreground region. The foreground region here was defined by conservative thresholding, wherein the threshold was determined as the overall mean intensity value for the whole scene.

Although the speed is still not really interactive, if affinity needs to be computed during the interactive process, in many cases, it is sufficient to do the computation and the tracking only on a 2D slice. This can be done in real time, allowing interactive parameter settings for the affinities and threshold selection. The connectivity scene (and the final object segmentation) can then be obtained for the whole 3D (or even 4D) scene in just a few seconds.

In the following sections, we present a comparison of the various tracking algorithms. All results include only tracking time. That is, the fuzzy affinities have been pre-computed for the whole volume before the tracking (and time measurement) took place.

4.2.3. Software — Tracking Without Preset Threshold

The chart in Fig. 2a shows the average tracking time (over the 10 MRI scenes) needed for the three objects using the different tracking algorithms. The algorithms are sorted in “goodness” order from left to right.

Our first observation is that the label-setting algorithms perform much better than the label-correcting variants, as we expected. The two exceptions are 1023 and 1021, where the choice of the hash function does not seem to be very useful.

These two hashing versions have very small hash tables, hence long lists for each bin. Further, 1021 puts voxels lying on the same (oblique) plane into the same bin, so big patches of neighboring voxels get into the same bin, further degrading hashing. However, the worst label-setting algorithm (besides the two extremes mentioned above) is still about three times faster than the best label-correcting algorithm. Further, there is a factor of five difference in the speed between the best and the worst label-setting algorithms.

As for the label-correcting versions (00xx), those using the optimal condition in Step 9 of Algorithm 1 (0002) are 20–25% faster than those with the condition in the previous (extensively used) implementation (0001),^{15,16} and 10–15% faster if using the binary map helper structure to avoid multiple instances of the same voxel in the queue (0012 and 0011). The additional memory needed for the binary maps (< 0.5 MB) is not really significant compared to the rest of the scene data we need to store (two input scenes (6.5 MB each), affinity values (3×6.5 MB), connectedness values (6.5 MB), altogether ≈ 39 MB).

The use of Fibonacci heaps (11xx) compared to binary heaps (10xx) seems to be slightly worse (mainly due to the more complicated data structure management and the sparseness and very simple structure of the graph in case), although in case of the additional direct pointers (1110 and 1010), the Fibonacci heap is faster. However, the memory storage required for these direct pointers is quite large (13 MB for the size of our scenes) — one third of the total other memory required — and the speed increase is only a few percent with the binary heap and approximately 20% with the Fibonacci heap, which itself is only slightly better than the corresponding binary heap.

The algorithms based on Dial’s implementation 12xx proved to be the best for computing all connectedness values. Using 1210 and pre-computed affinities, the tracking time for the connectedness values for WM, GM, and CSF objects is 4.72, 6.37, and 8.13 seconds, respectively, on average. This is certainly interactive time and is less than 1/3 of the time taken by the best heap algorithm to perform the same task. Although the “plain” versions (1200 and 1201) are always worse than 1210 (with the additional pointer-array, hence avoiding the search in the buckets), there seems to be a big difference as to how the FIFO and the LIFO versions perform for different objects. The FIFO version seems to be better than any of the non-Dial’s variants for the WM and GM objects, however, for CSF, it is worse than all “good” heap-based algorithms. This is due to the more complex and fine structure of particularly the peripheral CSF.

4.2.4. Software — Tracking With Preset Threshold

The chart in Fig. 2b shows a comparison of the algorithms in a way similar to that in the previous section except that, in this case, the algorithms made essential use of pre-determined thresholds for the fuzzy connectivity strength. The following changes are required to incorporate threshold into the algorithms. In Algorithm 1, Steps 5-9 should be executed only if $f_o(c) < x$, where x is the pre-determined threshold (see Algorithm $\kappa\theta_x FOE$ in¹⁵). In Algorithm 2, the while loop should stop after Step 4 when $f_o(c) < x$ for the first time. The thresholds were arrived at by averaging manually selected thresholds on the connectedness values for a few volumes and truncated on a 0.1 precision scale. The threshold for WM and CSF is 0.5, and that for GM is 0.7, on the $[0, 1]$ scale.

Note that the time axis in Fig. 2b has a range of about 20% of that in Fig. 2a. As in Fig. 2a, the drops corresponding to the label-setting algorithms (1xx) and the Dial’s-based algorithms (12xx), are significant. However, note also that the relative position of the curves has changed too: $CSF > GM > WM$ in the previous case, and $GM > WM \approx CSF$ with use of the pre-determined thresholds. Note also that the two hashing versions (1023 and 1021), that were extremely slow in the whole connectedness computation, are performing significantly better when the thresholds are known in advance, although they are still the worst among the label-setting algorithms. Surprisingly, the two algorithms using quite different hash functions (1020 and 1022) have almost the same performance when only the voxels belonging to the fuzzy objects get inserted into the heap. This was not true without using the thresholds. The few changes in the order among the other techniques are mainly due to the fact that less voxels get inserted into the queues/heaps/lists, and therefore, the search for the elements is more effective, and the cost of maintaining the data structure and the cost spent in the tracking loop cover different fractions of the total cost. Here too, the best algorithms are those using affinity-based hashing (12xx), with a performance almost three times better than that of the best heap-version. Also, the FIFO version is significantly better than any of the heap-versions in all cases.

The chart in Fig. 3 presents the tracking time spared when using some pre-determined threshold value x , as a function of x , as compared to the case with no preset threshold (i.e., $x = 0$). It is well illustrated here that even a small value of x (such as 0.1, which is significantly less than the final threshold) can save considerable amount of computation (30–50%). Data for the fastest tracking algorithm (1210) were used in this comparison. For inferior tracking algorithms, the effect of the small initial threshold is even more substantial (not demonstrated here).

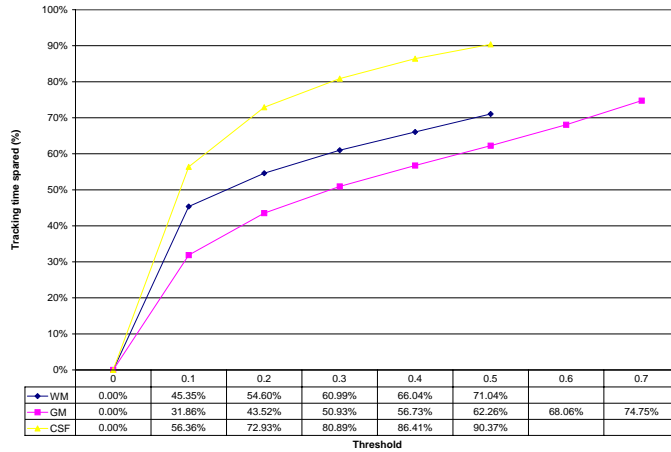


Figure 3. Comparing the average tracking time spared over the 10 MRI scenes when using different pre-determined thresholds for the fuzzy objects. The time spared is expressed in percent of the time needed for the complete tracking without a threshold.

5. CONCLUDING REMARKS

We have presented a family of 18 algorithms for fuzzy-connected object segmentation in volume images under two groups — label-correcting and label-setting — and under a variety of strategies of implementation for each group. We tested these algorithms in real medical applications, namely the separation of brain tissues in MRI in routine clinical trials. Changing the hardware only (from a SUN SPARCstation-20 to a PC with PentiumII processor) results in about 6 to 8 times faster implementations. However, this does not offer interactive operations. Utilizing efficient algorithms and careful selection of implementations can speed up the computation of fuzzy connectedness values by a factor of 20 or more (on the same hardware), as compared to the implementation previously used in our applications utilizing fuzzy object segmentation. The running time is further reduced considerably (by a factor of 3 to 10), when the algorithms make use of pre-determined thresholds for the strength of connectedness of fuzzy objects. In real applications, small thresholds can be used in advance without compromising the final outcome of the segmentation. This can save valuable seconds (in case of medium size images, such as the MRI brain images) and minutes (in case of large images, such as head or body CT), for real-time, interactive applications. In routine applications (such as MS lesion analysis¹⁶), the thresholds are known and hence can be preset, resulting in a tremendous saving in time.

We may conclude that there is no “always optimal” algorithm. The optimality of an algorithm depends on the input data as well as on the type of affinity relations used. However, some of the algorithms have definitely consistently better performance than others. The label-setting algorithms are faster in real applications. However, those using more complicated data structures (such as Fibonacci heaps rather than binary heaps) are not the fastest in practice. This is due to the big differences in the “constant” in their theoretical complexity formulas. Many analyses do not account for search for stored elements in queues/heaps/lists that can be eliminated only by large pointer-arrays, which may not be affordable (e.g., ≈ 100 MB additional memory in case of a CT scene).

Using the fast algorithms on fast (but not necessarily top-of-the-line hardware) interactive speed (about 5 seconds/scene) of fuzzy object segmentation is achievable. This leads to a new line of segmentation techniques. Utilizing the robust fuzzy connectedness principles and fast algorithms, interactive affinity parameter, seed point, and threshold selection becomes possible and the segmentation can be more efficient (in terms of both the user interaction time and computing time) without compromising the proven precision and accuracy of these techniques. If images having intensities with tissue specific meaning (such as CT or standardized MR images) are utilized, all parameters for the segmentation method can be fixed, fuzzy affinities and other intermediate data can be computed before the user interaction is needed and the user can be provided with more information at the time of interaction. More importantly, for very difficult segmentation tasks, perhaps user-steered techniques can be devised wherein fuzzy connected segments are collected within a 2D/3D/4D ROI swept by the user interactively within the scene domain.

ACKNOWLEDGMENTS

The authors’ work is supported by an NIH grant NS 37172 and a grant from the Department of Army DAMD 179717271.

The work of the second author (A.X.F.) is partially supported by CNPQ (Proc. 300698/98–4) and FAPESP (Proc. 98/0614–5 and 98/12308–8). The authors are thankful to Drs R. I. Grossman, D. C. Hemmy, and R. A. Baum for the data sets utilized in this research.

REFERENCES

1. L. Gong and C. A. Kulikowski, "Composition of image analysis processes through object-centered hierarchical planning," *IEEE Trans. Patt. Recogn. Mach. Intell.* **17**(10), pp. 997–1009, 1995.
2. M. Sonka, S. K. Tadikonda, and S. M. Collins, "Knowledge-based interpretation of MR brain images," *IEEE Trans. Med. Imaging* **15**(4), pp. 443–452, 1996.
3. J. C. Gee, M. Reivich, and R. Bajcsy, "Elastically deforming 3D atlas to match anatomical brain images," *J. Comput. Assist. Tomogr.* **17**(2), pp. 225–236, 1993.
4. W. L. Nowinski, "Dual probabilistic classifier for three-dimensional neuroimaging from MRI data," in *SPIE Proceedings*, vol. 2359, pp. 373–384, 1994.
5. N. R. Pal and S. K. Pal, "A review on image segmentation techniques," *Patt. Recogn.* **26**(9), pp. 1277–1294, 1993.
6. J. K. Udupa, S. N. Srihari, and G. T. Herman, "Boundary detection in multidimensions," *IEEE Trans. Patt. Recogn. Mach. Intell.* **4**(1), pp. 41–50, 1982.
7. H. K. Liu, "Two- and three-dimensional boundary detection," *Comput. Graph. Image Process.* **6**(2), pp. 123–134, 1977.
8. M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *Int. J. Comput. Vision* **1**(4), pp. 321–331, 1988.
9. S. Lobregt and M. A. Viergever, "A discrete dynamic contour model," *IEEE Trans. Med. Imaging* **14**(1), pp. 12–24, 1995.
10. T. McInerney and D. Terzopoulos, "Deformable models in medical image analysis: A survey," *Med. Image Anal.* **1**(2), pp. 91–108, 1996.
11. P. K. Sahoo, S. Soltani, A. K. C. Wong, and Y. C. Chen, "A survey of thresholding techniques," *Comput. Vision Graph.* **41**(2), pp. 233–260, 1988.
12. J. C. Bezdek, L. O. Hall, and L. P. Clarke, "Review of MR image segmentation techniques using pattern recognition," *Med. Phys.* **20**(4), pp. 1033–1048, 1993.
13. H. Rusinek, M. J. de Leon, A. E. George, L. A. Stylopoulos, R. Chandra, G. Smith, T. Rand, M. Mourino, and H. Kowalski, "Alzheimer disease: Measuring loss of cerebral gray matter with MR imaging," *Radiology* **178**(1), pp. 109–114, 1991.
14. R. Kikinis, M. E. Shenton, G. Gerig, J. Martin, M. Anderson, D. Metcalf, C. R. G. Guttmann, R. W. McCarley, W. Lorensen, H. Cline, and F. A. Jolesz, "Routine quantitative analysis of brain and cerebrospinal fluid spaces with MR imaging," *J. Magn. Reson. Imaging* **2**(6), pp. 619–629, 1992.
15. J. K. Udupa and S. Samarasekera, "Fuzzy connectedness and object definition: Theory, algorithms, and applications in image segmentation," *Graph. Models Image Process.* **58**(3), pp. 246–261, 1996.
16. J. K. Udupa, L. Wei, S. Samarasekera, Y. Miki, M. A. van Buchem, and R. I. Grossman, "Multiple sclerosis lesion quantification using fuzzy-connectedness principles," *IEEE Trans. Med. Imaging* **16**(5), pp. 598–609, 1997.
17. J. K. Udupa, D. Odhner, J. Tian, G. Holland, and L. Axel, "Automatic clutter-free volume rendering for MR angiography using fuzzy connectedness," in *SPIE Proceedings*, vol. 3034, pp. 114–119, 1997.
18. T. Lei, J. K. Udupa, P. K. Saha, and D. Odhner, "3D MR angiographic visualization and artery-vein separation," in *SPIE Proceedings*, vol. 3658, pp. 58–66, 1999.
19. J. K. Udupa, J. Tian, D. C. Hemmy, and P. Tessier, "A pentium personal computer-based craniofacial three-dimensional imaging and analysis system," *J. Craniofac. Surg.* **8**(5), pp. 333–339, 1997.
20. P. K. Saha, J. K. Udupa, E. F. Conant, and D. P. Chakraborty, "Near automatic segmentation and quantification of mammographic glandular tissue density," in *SPIE Proceedings*, vol. 3661, pp. 266–276, 1999.
21. B. M. Carvalho, C. J. Gau, G. T. Herman, and T. Y. Kong, "Algorithms for fuzzy segmentation," *Patt. Anal. Appl.* **2**(1), pp. 73–81, 1999.
22. P. K. Saha, J. K. Udupa, and D. Odhner, "Scale-based fuzzy connected image segmentation: Theory, algorithms, and validation," *Comput. Vision Image Understanding*, in press.
23. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows : Theory, Algorithms, and Applications*, chapters 4, 5, A. Prentice-Hall, Englewood Cliffs, NJ, 1993.
24. R. Dial, "Algorithm 360: Shortest path forest with topological ordering," *Communications of the ACM* **12**, pp. 632–633, 1969.
25. L. G. Nyúl and J. K. Udupa, "On standardizing the MR image intensity scale," *Magn. Reson. Med.* **42**(6), pp. 1072–1081, 1999.