Multi-Atlas Based Image Segmentation Applied on Pulmonary CT Scan Data

Maruti Agarwal

Department of Media and Knowledge Engineering
Delft University of Technology

Multi-Atlas Based Image Segmentation Applied on Pulmonary CT Scan Data

Maruti Agarwal

Department of Media and Knowledge Engineering
Delft University of Technology

Thesis Committee:
Prof. dr. ir. M.J.T. Reinders, EEMCS, TU Delft
Dr. B. C. Stoel, LKEB, LUMC
Dr. C. Botha, EEMCS, TU Delft
Dr. E. A. Hendriks, EEMCS, TU Delft
Dr. M. Staring, LKEB, LUMC

Contents

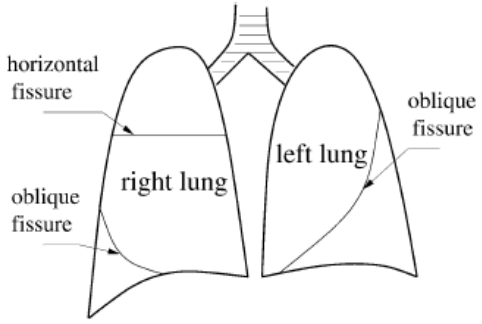
Chapter 1

Introduction

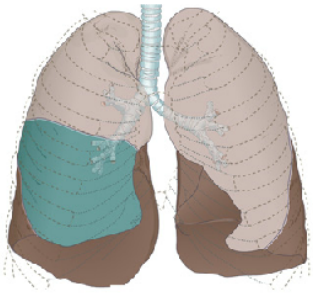
The development of an automatic system for analysis of the medical images is a well researched topic and it has attracted the attention of various researchers over the decades. The medical images such as X-ray images, Computed Tomography (CT) scans, magnetic resonance imaging (MRI) scans play a key role in the diagnosis and treatment planning of several diseases. In the case of lung diseases, chest X-ray and CT scan images are the first choice of medical practitioners for detecting abnormalities. X-ray images provide a $2D$ image of the chest, while CT scan is a more developed $3D$ version of X-ray. The chest CT scans can demonstrate various lung disorders such as lung cancer, tuberculosis, emphysema and chronic obstructive pulmonary disease (COPD). The chest CT scans are even able to demonstrate the nodules, vessels, and lobes in the lung.

1.1 Application Area

A typical image of human lung contains a right and a left lung. The right lung is divided into three lobes: upper, middle and lower as shown in figure 1.1(a). An oblique fissure separates the lower lobe from the middle, while a horizontal fissure separates the upper lobe from the middle. On the other hand, left lung is divided by an oblique fissure into two lobes: upper and lower. As can be seen in the figure 1.1(a), the oblique fissure is present in both the lungs while the left lung has no middle lobe, so there is no horizontal fissure on that lung.
A lung disease may affect different lung lobes independently. In fact; a disease may be confined to an individual lobe too. As an example, emphysema disease may be restricted within upper lobes in both the lungs. An emphysema patient is more likely to gain benefit from lung volume reduction surgery (LVRS) if the disease is located predominantly in the upper lobes emp [2003]. Lobe-by-lobe analysis of the lungs may not only be useful in treatment planning, but can also help in identifying COPD or interstitial lung diseases (ILD). Therefore, an automated scheme for lobe segmentation can be very helpful to medical doctors. In the current work, we aim to develop automated methods for the lobe segmentation.
As shown in figure 1.1(b), a lobe segmentation image contains different labels (or colors) for separate lobes. The goal of this research is to do lobular segmentation for pulmonary (chest) CT scan data. Since fissures are boundaries between the lobes in the lungs, the task of lobe-segmentation can also be approached by *fissure detection*. One straightforward way of fissure detection is manual delineation but manual delineation requires expert knowledge. It is very tedious and time consuming. The reproducibility of manual delineation can be questioned from the fact that different people mark slightly different fissures for the same patient data (inter-



(a) Adopted from Zhang et al. [2006].



(b) An example of lobe segmentation. Separate lobes are shown in different colors.

Figure 1.1: Lung lobar anatomy. Left lung has two lobes separated by an oblique fissure; the right lung has three lobes, with the upper and middle lobes separated by a horizontal fissure and the middle and lower lobe separated by an oblique fissure.

observer differences). The differences are observed even when the same person delineates the fissures manually (intra-observer differences) at different points in time. The current work is presented as an alternative to the manual fissure detection.

1.2 Atlas Based Segmentation Framework

There are several automatic and semi-automatic approaches for the lobe segmentation. We employ a specific atlas-based segmentation (ABS) framework for this task since it is more suitable for our application area than intensity based segmentation approaches. The details on this are provided in chapter 2. An atlas is a set of two images: an intensity image and its segmentation. Figure 1.2 shows an example atlas for CT scan data of the left lung. The anatomical information in the atlas is used to obtain a segmentation of an unseen image. ABS is a two step process:

1. Intensity image in the atlas is transformed to make it similar to the unseen image. This gives us a transformation from the atlas domain to the unseen image domain. See figure 1.3(a).

2. This transform is applied on the segmentation image in the atlas. The resultant image obtained through this transformation is a segmentation of the unseen image. See figure 1.3(b).

It should be noted that the accuracy of the segmentation of the unseen image would depend on the accuracy of the transformation performed in the first step. If the transformation from the atlas to the unseen image is perfect, then we would obtain a perfect segmentation of the unseen image.
The transformation from the atlas to the unseen image in the first step is performed using *image registration*. In image registration, an image is deformed to make it look similar to another image. In other words, one image is transformed to another image. The intensity image in the atlas, called as moving image, is transformed to the unseen image (called as fixed image), so that both the images appear to be similar. The accuracy of the first step in ABS, i.e. image registration, is crucial for obtaining a good segmentation of the unseen image. The

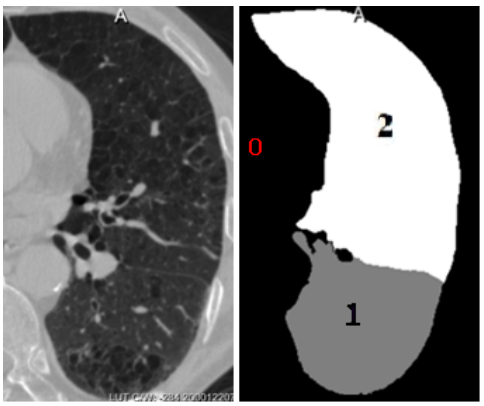
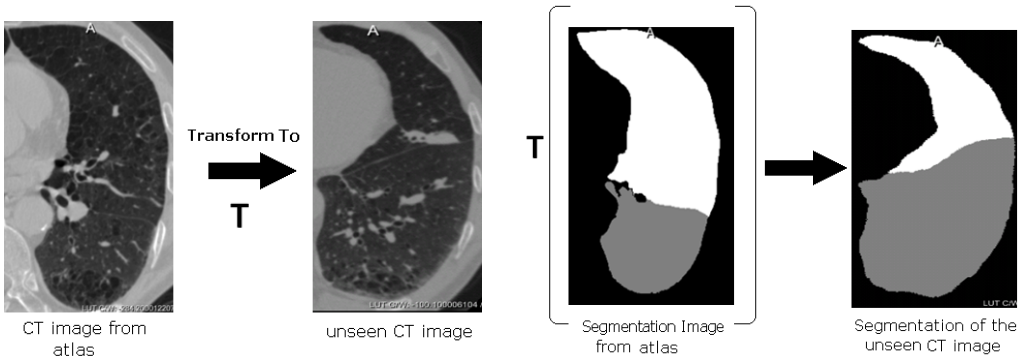


Figure 1.2: An example atlas of the left lung: (Left) CT scan image ; and (right) segmentation of the CT scan image. The segmentation of the left lung contains: label 2 for the upper lobe, label 1 for the lower lobe, and label 0 for the background.



(a) First step in ABS.

(b) Second step in ABS.

Figure 1.3: Atlas based segmentation. In the first step, the intensity image from atlas is transformed to the unseen image. In the second step, the transformation obtained from the first step is applied on the segmentation image in the atlas.

inaccuracies in the registration may cause a poor segmentation from the output of the second step in ABS. One way of coping with this would be to use multiple atlases, which is discussed in the next section.

1.3 Multi-Atlas Based Segmentation

Suppose we are provided with a set of 4 atlases and an unseen image. These atlases can be transformed to the unseen image domain in four independent ABSs. This results into four segmentations of the same unseen image. Considering the transformation (or registration) to be imperfect in all the four ABSs, these four segmentations may not look exactly the same. Figure 1.4 shows a similar case where the four poor segmentations of the unseen image are obtained. But if we give a closer look to these fours segmentations, then we realize that they are *locally correct* in different regions. These locally correct fissures in the four segmentations are highlighted in the figure 1.4. In the current work, the focus of *multi-atlas based segmentation* approach is on picking up these locally correct *red* fissures from the four segmentations and combining them to form a good estimate of the *green* fissure of the unseen image. This task

of combining *relevant* information from various segmentations can be termed as 'segmentation fusion'. As long as the registration errors in the independent ABSs are non-symmetric, it is likely that the local errors in the various segmentations of the unseen image can be corrected by combining these segmentations.

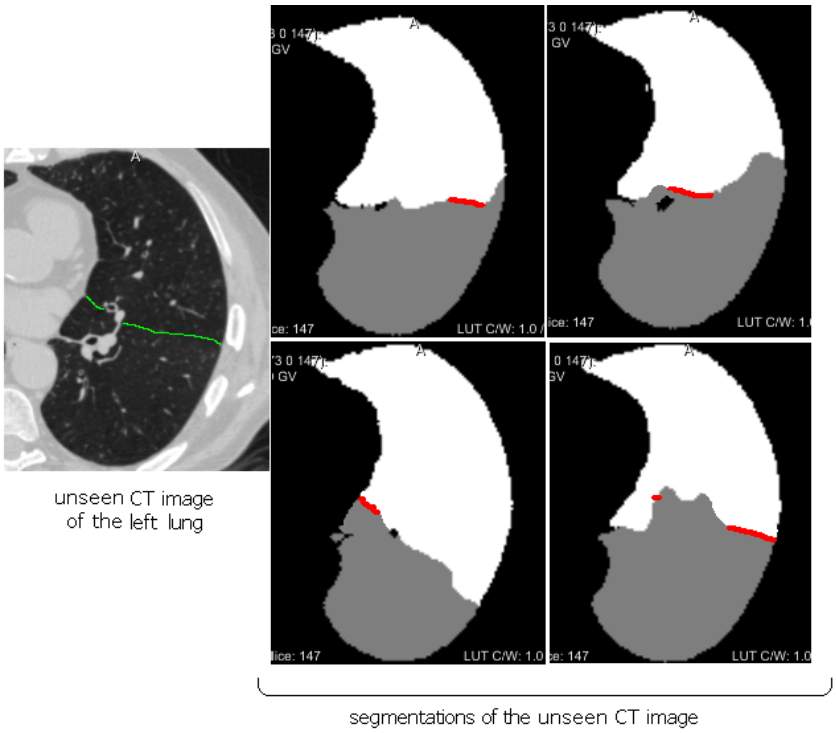


Figure 1.4: An unseen CT image of the left lung and its four segmentation are shown. Fissure is highlighted in the unseen image and the corresponding *matching* part of the fissure is highlighted in the four segmentations. We focus on picking up these locally highlighted regions from the segmentations and combine them to obtain entirely correct fissure for the unseen image.

1.4 Research Goals and Contribution

The goal of current research is to detect the lobes in the pulmonary CT scan images. We use a specific multi-atlas based segmentation (MABS) approach which requires the 'segmentation fusion' step for obtaining a segmentation of the unseen image. The strategy for fusion of the various segmentations should focus on selecting 'locally-correct' regions from each segmentation. We studied the effect of the success of registration (or ABSs) on the output of the segmentation fusion. Furthermore, the report contains several novel algorithms for segmentation fusion. One previously published approach to the segmentation fusion: selective and iterative method for performance level estimation (SIMPLE) performed well on our dataset. Some of our proposed algorithms obtained significant improvement over the performance of SIMPLE. Some of the algorithms developed in the current work can be used in any application area for combining various segmentations, while some parts of the algorithms are specifically for lobular segmentation of the pulmonary CT scan data.

1.5 Thesis Outline

In this chapter, we discussed the motivation of this research, application area, a brief introduction of the atlas based segmentation framework, and research goals.
Second chapter provides a literature review of various research attempts relevant to ABS. A detailed review of ABS approaches with discussion on their merits and demerits is provided. Furthermore, various approaches to segmentation fusion are discussed with a closer look at some of the popular approaches. Finally the chapter ends with a conclusion.
In the third chapter, we have provided details of our pulmonary dataset, softwares used in current work, and registrations setup. We have experimented with two different registration setups, let's call them A and B. Since the accuracy of registration, i.e. step-1 in ABS, directly affects the segmentation output from the second step, we studied the effect of the two registration strategies on the accuracy of the segmentation output. Both the registration approaches are discussed in this chapter. Finally, the format in which we have reported our results is described.
Fourth chapter is focussed on a specific method for segmentation fusion: SIMPLE. SIMPLE is found to be the best performing algorithm on our dataset compared to several existing segmentation fusion algorithms. We have proposed several modifications in SIMPLE. Some of the modifications provide significant improvement over SIMPLE. We have presented results with detailed analysis for each of our proposed algorithm. All the results presented in this chapter are obtained on using registration setup-A for all the ABSs. This chapter contains results only for the left lung. The chapter ends with concluding remarks.
Fifth chapter contains results of various segmentation fusion algorithms proposed in the fourth chapter. But this chapter makes use of registration setup-B for performing ABSs. The results are presented for both the left and the right lung. Since accuracy of registration is crucial to the output of a segmentation fusion strategy, in this chapter we have focussed on studying the impact of image registration on the segmentation fusion output quality. The setup-B is considered an improvement over the setup-A. It is quite interesting to see how drastically the output of segmentation fusion is affected by the registration accuracy.
In the sixth chapter we make direct use of registration accuracy for identifying the locally good regions from the segmentations. The registration from the atlas to the unseen image may be locally correct. We use this local correctness of registration as criteria to select the locally good parts of the segmentations. In addition to this, we also employ certain fusion strategies, called 'fixed rules', for combining the local regions within the segmentations. Therefore this chapter not only focuses on locating correct regions within the segmentations but also on combining these regions using several criteria. These fixed rule fusion strategies have been used in the literature for the task of combining pattern classifiers. The motivation of employing the fixed rules comes from the analogy between the problem of segmentation fusion and pattern classifier combination. This chapter also contains details of the ideas which have not worked well but may serve as motivation for future research. The chapter also contains a discussion on the results and then it concludes.
Seventh chapter discusses the ideas which have not performed to the expectation. Since an insight into these ideas may help future efforts, we have described the motivation behind them. A final discussion, conclusion and scope for future work has been provided in the chapter eight.

Chapter 2

Literature Review

2.1 Notations Used in Current Work

Table 2.1: Notations

Symbol	Name
x	a voxel position
M_i	i^{th} moving image
S_i	Manual segmentation of M_i
F	Fixed image
DM_i	i^{th} deformed moving image
DS_i	i^{th} deformed segmentation image
GT	Ground truth, i.e., manual segmentation of F
$E(GT)$	Estimated ground truth
$dist_{M_i}$	Distance map for the moving image M_i
$dist_F$	Distance map for the fixed image F

This chapter provides an overview of the available literature on the atlas based segmentation (ABS) approaches for finding the segmentation of an unseen image. A brief introduction of the image registration is also provided to help in understanding the details of ABS. There have also been several research attempts to combine the information from multiple atlases in order to improve the accuracy of the segmentation of the unseen image. Such multi-atlas based segmentation (MABS) approaches have been proven to outperform the single ABS approach. In fact the MABS approach which makes use of the *segmentation fusion* is found to be more accurate as compared to other multi-atlas based methods. The chapter gives more insight on this discussion. An overview of the literature relevant to the segmentation fusion is also provided. Finally this chapter concludes with an explanation of our approach, for the current work.

2.2 Atlas Based Segmentation

There are several intensity based classification methods for image segmentation. Usually such methods assign a label to each voxel based on its intensity value. However, there are certain applications where it is not possible to establish a well-defined relation between the intensity of

a voxel and its label, e.g. pulmonary CT scan data. The upper and lower lobes in the left lung exhibit a similar range of intensity values. Therefore it is not possible to distinguish both the lobes by using only their intensity values. What differentiates the upper lobe from the lower one is its spatial location and a fissure separating both lobes (figure 1.1(a)). Thus, the problem of finding fissure in a left lung image can also be seen as segmenting the lung into the upper and the lower lobes. Similar argument holds for the right lung too, where detection of the oblique and the horizontal fissure can be used to separate the three lobes.
Suppose we are provided with a CT image of lung and its segmentation. This set of two images, called an atlas, not only gives us anatomical information of lungs but also provides useful information about the fissure such as its structure and location in the lung region. The knowledge about the fissure can be used to obtain the fissure in an unseen CT image. The ABS approach makes use of this knowledge to find the fissure in the unseen image by finding a correspondence between the atlas and the unseen image. Since ABS makes use of image registration, it is important to understand basics of registration to get a good understanding of ABS process.

2.2.1 Image registration

This subsection is adopted from Klein and Staring [2004]. Although the image registration process is not the main focus of our work, a brief introduction is provided for basic understanding. Image registration is the task of aligning two images into a common coordinate system. One image, called *moving image* M, is transformed into the coordinate system of the other image, called *fixed image* F. Registration is the problem of finding a transformation $\boldsymbol{T}$ which can spatially align the moving image M with the fixed image F. Therefore, the registration process tries to find a displacement $u(x)$, for a voxel x, that makes $M(x+u(x))$ spatially aligned to $F(x)$. In other words, registration process involves finding a transformation $T(x) = x + u(x)$ that makes $M(T(x))$ spatially aligned to $F(x)$. Commonly, the registration problem is formulated as an optimization problem in which the cost function $\mathcal{C}$ is minimized w.r.t. $\boldsymbol{T}$:

$$\hat{\boldsymbol{T}} = \arg\min_{\boldsymbol{T}} \mathcal{C}(\boldsymbol{T}; F, M), \tag{2.1}$$

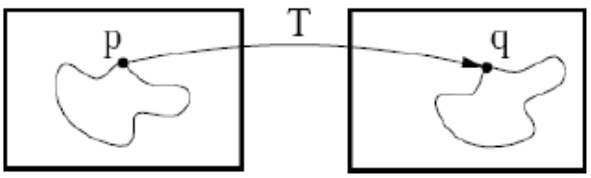


Figure 2.1: Image registration is the task of finding a spatial transformation mapping one image to another. Adopted from Ibáñez et al. [2005]

The transformation model used for $\boldsymbol{T}$ determines what type of deformations between the fixed and moving image you can handle. In order of increasing flexibility, these are the translation, the rigid, the similarity, the affine, the non-rigid B-spline and the non-rigid thin-plate spline like transformations. The equation 2.1 also has a cost function $\mathcal{C}$, or *metric*, which defines the quality of alignment. There are several choices for this metric such as: sum of squared differences (SSD), normalized correlation coefficient (NCC), mutual information (MI), and normalized mutual information (NMI). If the moving image M is perfectly aligned with the fixed image M after

the registration, the SSD cost function would be minimum. Refer Klein and Staring [2004] for details on this topic. Interested readers may also refer Klein et al. [2007], Maintz and Viergever [1998], Lester and Arridge [1999], Zitová and Flusser [2003], and Hill et al. [2001] for more literature on the image registration.

2.2.2 Image registration for atlas based segmentation

In the image registration terminology, the intensity image in the atlas is called as the moving image M and the unseen image is termed as the fixed image F. We have represented the segmentation image in the atlas by S. As we discussed in the chapter 1, ABS is a two step process:

1. The moving image M is registered to the fixed image F. This results into a transformation $\boldsymbol{T}$ from M to F. See figure 2.2.
2. The transform $\boldsymbol{T}$ is applied on the segmentation image S in the atlas. See figure 2.3.

This second step of transforming S is also known as *label-propagation*. On applying the transform $\boldsymbol{T}$ on S, we obtain a deformed segmentation image DS which represents the segmentation of the fixed image F.

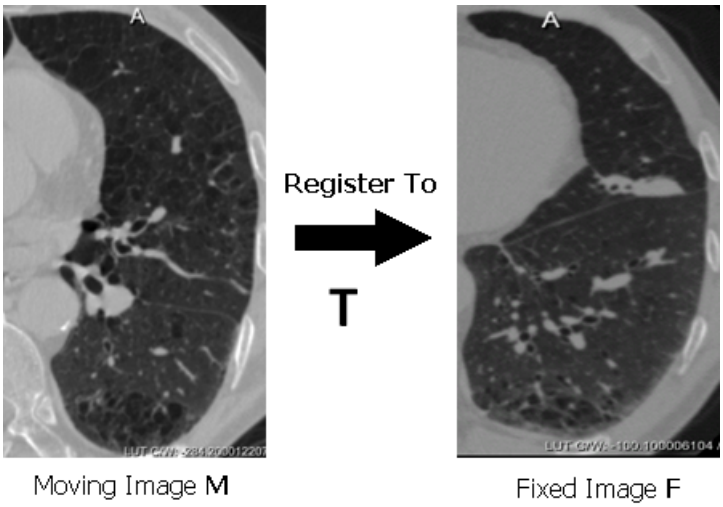


Figure 2.2: Moving image M is registered to fixed image F. As a result of this registration a transform T from the moving image to the fixed image is obtained.

Image registration makes it possible to separate one lobe from another, if we know the lobular segmentation for an observed lung image. The lobes for the observed image may be obtained by manual delineating the fissures.
In this report, we have interchangeably used the term moving image M to represent the CT scan intensity image of an atlas and fixed image F represents the unseen CT image. The segmentation in the atlas is denoted by S which represents the segmentation of the moving image M. The deformed segmentation image DS obtained from the second step in ABS is called as an *estimate* of the segmentation of the fixed image F. The actual segmentation of the fixed image may be found by manual segmentation. It is called as *ground truth (G)*.

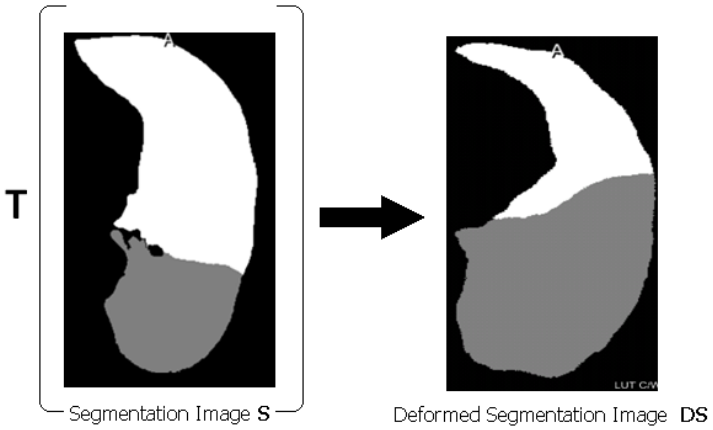


Figure 2.3: Transform T obtained from the previous registration step is applied on the segmentation S of the moving image M. Deformed segmentation DS is obtained. DS is an estimate of a segmentation of the fixed image F.

2.3 Using More Than One Atlas

We can obtain an estimate of a segmentation of the fixed image by a single ABS. There have also been several attempts to combine information from several atlases into one atlas. The motivation of such attempts is to combine information from many atlases into one atlas. This one atlas is expected to be more informative that any individual participating atlas. There are three famous approaches which try to converge to a single atlas starting from multiple atlases:

1. Creating an average atlas
2. Creating a probabilistic atlas
3. Using a single best atlas

2.3.1 Creating an average atlas

In Guimond et al. [2000] and Rohlfing et al. [2001], a method to generate an 'average atlas' is described. The average atlas tends to possess variability of all the individual atlases. This average atlas contains two important properties: an average intensity and an average shape, both in a single image. The average model is obtained by deforming an average image. This average image is obtained from the deformed moving images which are registered to a common fixed image. The deformation applied on this image is computed by taking inverse of the average of all the individual deformations as shown in figure 2.4. Rohlfing et al. [2001] demonstrated that the use of such average atlas can provide better segmentation result than any individual participating atlas.
In Stancanello et al. [2006] and Wu et al. [2006] several atlases are obtained from the *same* patient, i.e. intra-patient data. One of the images is selected as fixed image and remaining images are independently registered to the fixed image. Then an average atlas is created by using all the deformed moving atlases. In Carmichael et al. [2005], Sluimer et al.,Zhang et al. [2006], and Li et al. [2003] an average atlas is created by using multiple images from *different* patients, i.e. inter-patient data.
The advantage of such approach is that once an average atlas is generated, only a single registration from the atlas to the unseen image is required to obtain its segmentation. This involves

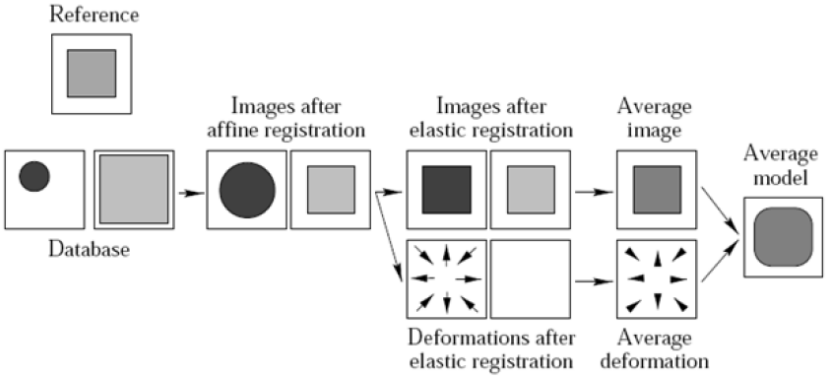


Figure 2.4: Average atlas construction. Adopted from Guimond et al. [2000]

much less computation as compared to performing separate ABS for each atlas. There are two problems associated with using an average atlas as mentioned in Rohlfing et al. [2005]:

1. An average atlas may not be a good representative of all the participating atlases.
2. The resultant segmentation depends completely on one registration from the average atlas to the unseen image. If this registration goes wrong, we practically lose the advantage of having multiple atlases.

2.3.2 Creating a probabilistic atlas

Information from several atlases can be combined into an average atlas containing an intensity value for each voxel. Each voxel in the segmentation image of this atlas contains a crisp label which shows the segmentation decision for this voxel. Instead of such crisp labels, one can also find probability with which the voxel belongs to a particular label. Combining multiple atlases to form a probabilistic atlas would give such a result. A review of literature on probabilistic atlas can be found in Park et al. [2003], Sluimer et al., Lorenzo-Valds et al. [2004], and Svarer et al. [2005].
The probabilistic atlas does not just mean the average boundary of any shape. It comprises complete spatial distribution of probabilities, e.g., in our application, it would represent the probability that a voxel in left lung image belongs to the upper, the lower lobe, or the background. Therefore each voxel would be represented by a n length vector, where n is the number of different crisp labels present in the segmentation images of atlases. For the left lung n would be 3, while for the right lung n is equal to 4. Once a probabilistic atlas is registered to the unseen image, one can find the most probable label for any voxel in the unseen image.
The limitations of probabilistic atlas are same as that of an average atlas. Lack of representation of the entire population and too much dependency on a single registration are the factors which motivate researchers to find alternate ways.

2.3.3 Using single-best atlas

In Rohlfing et al. [2005], two approaches are proposed to choose the *best* atlas from a given set of atlases. In the first approach, one of the atlases is chosen manually. The choice of atlas was based on visual assessment of image quality and intensity uniformity. In the second approach, the atlas which is most similar to the unseen image is chosen. In this case the choice of atlas was based on four pre-defined criteria:

1. Value of normalized mutual information (NMI) after affine registration (NMIaffine). Each atlas is registered to the unseen image using affine transform. NMI of the transformed atlas and unseen image is computed and the atlas with highest NMI is chosen.

2. Value of NMI after nonrigid registration (NMInonrigid). As the name suggests, this criteria makes use of non-rigid registration and selection criteria is same as that of the NMIaffine.

3. Average deformation of the atlas over all voxels (DEFavg). After nonrigid registration, the deformation between an atlas and the unseen image is computed and averaged over the number of voxels. Atlas with least average deformation is selected.

4. Maximum deformation of the atlas over all voxels (DEFmax). This criteria is same as DEFavg, except that it makes use of maximum deformation instead of average.

Based on the results described in Rohlfing et al. [2005], NMInonrigid criteria worked best. In all of these single-best ABS approaches, final segmentation relies on the success of only one registration.

2.4 Multi-Atlas Based Segmentation

MABS methods tend to outperform single-ABS methods , Rohlfing et al. [2004a], Klein et al. [2005], Svarer et al. [2005], and Heckemann et al. [2006]. Rohlfing et al. [2004a] demonstrated that in order to obtain the full advantage of using multiple atlases, one should combine the segmentations of individual deformed atlases, i.e., register each moving image M to the fixed image separately and combine all the deformed segmentations obtained from independent ABSs. Assume N moving images $M_i(i = 1, 2, 3 \cdots N)$ and their corresponding segmentations $S_i(i = 1, 2, 3 \cdots N)$. Consider a three step process for obtaining the segmentation of the fixed image F:

1. each M_i is registered to the fixed image F independently, resulting in transforms T_i.

2. The segmentations S_i are transformed by applying T_i transformations. As a result, we get total N estimates $DS_i(i = 1, 2, 3 \cdots N)$ of the segmentation of the fixed image F.

3. Combine the segmentations $DS_i(i = 1, 2, 3 \cdots N)$ to obtain a unique segmentation decision for the fixed image F.

The third step is termed as *segmentation fusion* or *label fusion*.

2.4.1 Atlas Selection

In literature, various heuristics have been proposed for selecting only a subset of the atlases from all the available atlases. One can apply the segmentation fusion based approach on the set of selected atlases in order to reduce the computational burden of performing registrations for all the atlases. Atlas selection alleviates the problems associated with the usage of an average atlas, probabilistic atlas, or single-best atlas. Therefore, the advantage of atlas selection approaches are two-fold:

1. The segmentation of the unseen image does not depend only on a single registration but on the registrations of the selected atlases.

2. It is computationally less expensive to register the selected atlases instead of registering all the available atlases.

One can select a subset of atlases either *before* performing registration Aljabar et al. [2009], Rohlfing et al. [2004a] or *after* the registration Klein et al. [2008], Artaechevarria et al. [2009], I. Isgum and van Ginneken [2009], and van Rikxoort et al. [2010b]. Although pre-registration atlas selection can be used to reduce the number of atlases before the actual ABS procedure, a post-registration atlas selection is found to be more accurate in practice. Since the registered atlases contain more useful information about their similarity with the unseen image. This makes it easier to select the atlases which are most similar to the unseen image. In a similar attempt Klein et al. [2008] performed a coarse registration in first and the used the similarity of its output with the unseen image as a criteria for atlas selection. Next, a finer registration was performed only for the selected atlases. This methodology gives advantage of post-registration atlas selection while it is computationally less expensive compared to normal post-registration atlas selection methods.
One can apply the *segmentation fusion* based approach on the set of selected atlases only in order to reduce the computational burden of performing several registrations. The segmentation fusion approach is discussed in more detail in the

2.4.2 Analogy with combining pattern classifiers

In this report, multi-atlas based segmentation is treated as a *segmentation fusion* problem. Another way of looking at this set up is by using pattern recognition terminology. The problem of segmentation fusion is analogous to the task of *combining pattern classifiers* Kittler et al. [1998]. We can consider each deformed moving atlas as a base classifier and the segmentation by an atlas as its classification decision. This way the task of combining several segmentations is analogous to combining the classification decisions from various atlases. Therefore, label fusion (or segmentation fusion) can also be seen as *combining classifier*. The two most common approaches for combining classifier are majority voting and weighted voting.

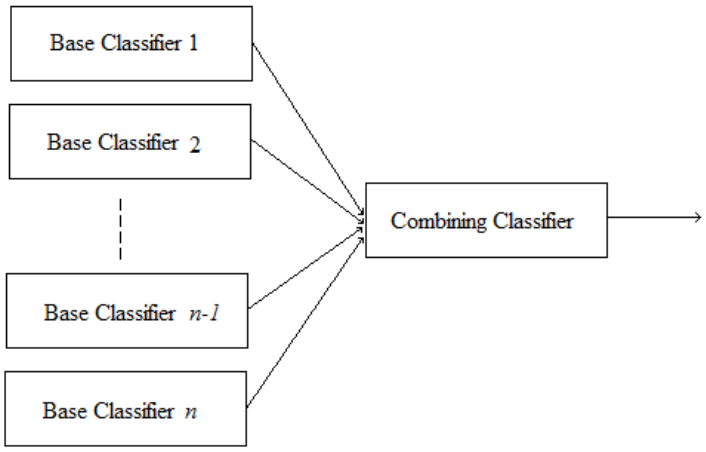


Figure 2.5: The decisions from several base classifiers (i.e. deformed atlases) are combined using a combining classifier (segmentation fusion).

2.4.3 Voting

Voting is the most intuitive and the simplest way of combining the decisions of the several observers. Consider these three decision making rules: unanimity, majority voting, and plurality voting. See figure 2.6. These rules are commonly used as *combining classifier strategies*. Lets assume black, white and gray colors shown in the figure 2.6 correspond to three separate classes. Each row represents 10 objects, which are already classified into one of the three classes. The final decision for the first row is black, on using *unanimity* for making the final decision. The unanimity requires 100% support in the favor of the winning class. The final decision for the second row is black, on using the majority voting. Majority voting requires the winner class to have more than 50% votes. The ties are resolved arbitrarily. There is one more voting rule called as *plurality* voting. A class with maximum support is takes as winner according to the plurality voting. For the third row, shown in the figure 2.6, the decision of plurality voting would be black. It should be noted that majority voting cannot make any decision for this case because of the lack of more than 50% votes in the favor on one class. The final decision for all the three rows is black by the unanimity, majority voting, and plurality voting respectively.

Figure 2.6: 1^{st} row: Unanimity (all agree); 2^{nd} row: Simple majority voting (more than 50% votes) ; 2^{nd} row: plurality voting (maximum votes)

Plurality voting is often used with the name majority voting in the literature. To keep our terminology in accordance with the available literature, we have also used the term majority voting in this report, which actually represents plurality. Several researchers have used majority voting Aljabar et al. [2009], Heckemann et al. [2006], Rohlfing et al. [2004a], and Rohlfing and Maurer [2007] for segmentation fusion in the context of MABS. It makes use of very less information and only looks at the labels of the corresponding voxel x in the input segmentations DS_i in order to to decide the label for a voxel x in the fusion output. The main advantage of majority voting is its simplicity and low computation time.

2.4.4 Weighted voting

Weighted voting is a more complex combining classifier strategy than majority voting. In weighted voting, each input segmentation $DS_i(i = 1, 2, 3 \cdots N)$ is assigned some *weight*. This weight represents the confidence over the input segmentation. Therefore it is always a probability value between 0 to 1. An input segmentation DS_i may have a global weight. See figure 2.7 where a global weight of 0.8 is assigned to the entire image. Therefore whatever be the label of a voxel (0, 1 or 2), it has 0.8 confidence over its label. Instead of having one *global weight* value, each voxel in the image may have a *local weight* value. This local weight represents the

confidence over the label of that voxel. In this setting, an image needs to have as many local weight values as the number of voxels in it, one weight value for each voxel's label.

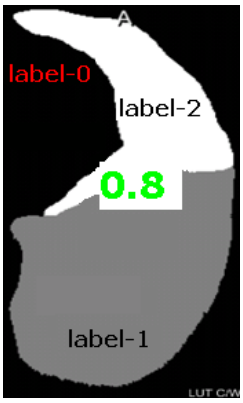


Figure 2.7: Global weights: a global weight value of 0.8 is assigned to all the voxels. Therefore each voxel has 0.8 confidence over its label.

Majority voting assigns default global weight 1 to all the input segmentation $DS_i(i = 1, 2, 3 \cdots N)$. If all the inputs are not equally good, then it is reasonable to give more weight to more accurate segmentation in making the final decision Artaechevarria et al. [2009], I. Isgum and van Ginneken [2009], Sdika [2010], and Rohlfing et al. [2004a]. In figure 2.8, the more accurate segmentation is assigned a higher weight value compared to the less accurate one.

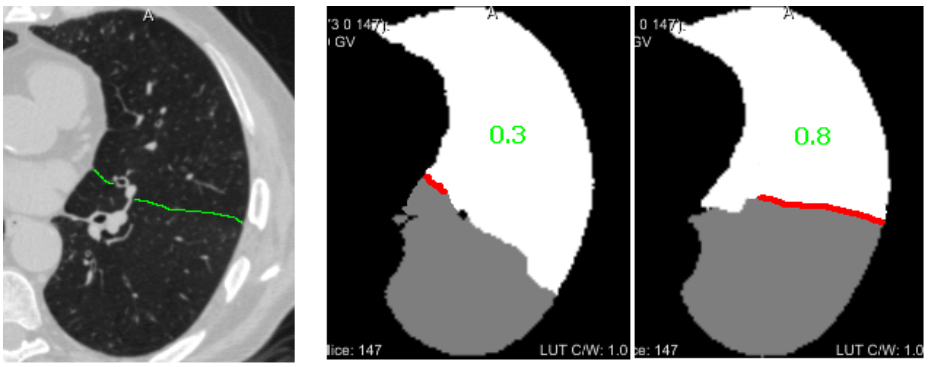


Figure 2.8: A higher global weight of 0.8 is assigned to the more accurate estimate of segmentation. Less accurate estimate gets lower weight value.

Similar to majority voting, weighted voting is also performed voxel-wise. For each voxel, we look at the total weight in the favor of the label of that voxel. N input segmentations $DS_i(i = 1, 2, 3 \cdots N)$ provide us N decisions for an output voxel x. In other words, each input segmentation contains a weight value for the label of voxel x We count the total weight in the support of each candidate label for voxel x. A label with the highest total weight is considered winner. The decision is made for each voxel independently. If weights are correct, one can expect weighted voting to perform better than majority. Finding reliable weights of individual segmentations is crucial to the performance of weighted voting. Various authors have introduced several approaches to estimate these weights. We have categorized these approaches in two different ways (figure 2.9):

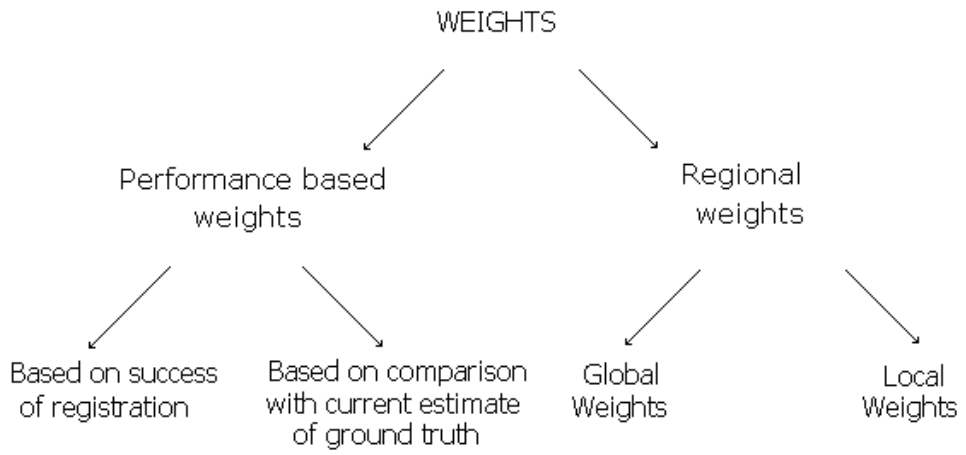


Figure 2.9: Weight Classification

1. **Performance Based Weight Calculation:**Performance based weights are further subdivided into two categories: (a) performance based on an estimate of ground truth; (b) performance based on success of registration. The approaches within both the categories are discussed in detail in this section.

2. **Regional Weights:**These type of approaches to weight calculation are divided into two categories: (1) Global weights; and (2) Local weights. In case of global weights, there is only one weight value for an entire input segmentation. This weight is same for all the voxels in the image. In case of local weight calculation, a weight value for each voxel in each input segmentation is computed. In some applications, such local weight based segmentation fusion approaches have been proved to attain higher accuracy as compared to the global weight based approaches. But there is also a possibility that the local weights may be severely affected by the presence of noise in the image data which may hamper the accuracy of local weight based fusion approaches. A more detailed discussion on this is provided in this section.

Remember that a deformed segmentation image DS_i obtained from the ABS is called as an *estimate* of the actual segmentation of the fixed image F, i.e. an *estimate of the ground truth*. While the final output segmentation obtained from the segmentation fusion of all the input segmentations $DS_i(i = 1, 2, 3 \cdots N)$ is called as *estimated ground truth*. This is the typical terminology used in the literature, therefore we employ the same to be in consonance with the existing literature.

Performance Based on an Estimate of Ground Truth: Usually approaches in this category make use of Expectation-Maximization (EM) algorithm. EM is a two step iterative process, which repeats expectation and maximization steps. The process starts with computing an estimate of the ground truth using a simple approach such as majority voting. Based on this estimate, it is possible to calculate the performance of individual segmentations DS_i. In expectation step (E-step), given the performance, input segmentations are combined to make an estimate of ground truth. Therefore, this step gives us an updated estimate of the ground truth. In maximization step (M-step), given an estimate of the ground truth, the performance of individual input segmentations is maximized. In this step, the performance values of input segmentations are re-evaluated. Generally, the performance is dependent on certain parameters and M-step focuses on finding the parameters which maximize the performance of the individ-

ual segmentations. In E-step, the estimate of the ground truth is improved, based on these parameters. This iteration continues till the process converges. One popular such approach is the Simultaneous Truth and Performance Level Estimation (STAPLE) method proposed by Warfield et al. [2004]. This method calculates weights for each voxel in the E-step. Furthermore, the sensitivity and specificity of each segmentation is calculated in M-step. Therefore, E-step computes a local weight value for each voxel in the each input segmentation, and M-step computes global performance measure for each input segmentation. This global performance measure is the sensitivity and the specificity of the input segmentations.
Warfield et al. [2004] and Rohlfing et al. [2004b] both proposed multi-label STAPLE. The method described by Rohlfing et al. [2004b] has the benefit that less memory is required since weights are not calculated for each voxel, but a confidence matrix for each segmentation is estimated in the E-step. The M-step remains similar to the original STAPLE algorithm Warfield et al. [2004], which focuses on improving the estimate of the ground truth. Therefore Warfield et al. [2004] makes use of confusion matrix based global weights.
In another approach called as SIMPLE, Langerak et al. [2010] used overlap between an input segmentation and the estimate of ground truth to find the weight for each input segmentation. The overlap measure used is DICE similarity coefficient. SIMPLE and STAPLE (as described in Rohlfing et al. [2004b]), both the methods start with computing an initial estimate of ground truth. First of all, the performance of each input segmentation is computed w.r.t. to this initial estimate and then this estimate is updated based on the performance found. Both the algorithms make use of global performance measure. They use an iterative approach in which the estimate of ground truth is improved over the iterations. The global weights for each input segmentation are updated in the each iteration based on the comparison with the current estimate of the ground truth. We obtain a final *estimated ground truth* as the process converges. As is mentioned in Langerak et al. [2010], STAPLE has been reported to outperform majority voting in Rohlfing et al. [2004b], but in some publications it has been shown that STAPLE does not give any significant improvement over majority voting Artaechevarria et al. [2009]. SIMPLE is reported to outperform majority voting, STAPLE, and several other local weights based fusion approaches mentioned in Artaechevarria et al. [2009].
Performance Based on Success of Registration: The approaches in this category usually rely on the deformed moving images DM_i and the fixed image F for weight calculation. They do not make any use of input segmentations DS_i for weight calculation. If the registration is perfect, a deformed moving image should be exactly the same as the fixed image. In such an ideal case, the weight for the segmentation obtained from this ABS should be 1. The idea behind such approaches is that an input segmentation obtained in the second step of ABS is only as much reliable as the registration from the moving image M_i to the fixed image F in the first step of ABS. Artaechevarria et al. [2009] used normalized correlation coefficient (NCC), normalized mutual information (NMC), and mean square distance (MSD) between the deformed moving image and the fixed image to calculate both: the global and the local weights. It was found that MSD based weights perform better than majority voting and STAPLE. In another work, I. Isgum and van Ginneken [2009] employed the absolute difference between deformed moving image and fixed image for weight calculation.
This completes the literature review of performance based weight calculation methods. Next, we study the regional weights, which are further divided into two sub-categories: global and local.
Global Weights: In case of global weights, there is a unique weight value, between 0 to 1, for an entire input segmentation. This means that all the voxels of S_i get the same weight. One

problem with global weight is that all the local regions within an input are assigned equal weight irrespective of their accuracy. Although, computation of global weights is computationally less expensive that local weights. E.g. the average size of an image (one lung only) used in our work is $250 \times 350 \times 600$ voxels. Therefore, total number of voxels is in the order of 1×10^7. Computing local weights for each voxel would be quite expensive. Global weights have shown reasonable accuracy in many researches such as Langerak et al. [2010] and Warfield et al. [2004]. The difficulty in calculating the local weights reliably makes global weights an attractive option. Local weights are reported to be sensitive noise Langerak et al. [2010].

Local Weights: Local weights take advantage of the fact that registration may be locally good in some areas while worse in others. As was mentioned, the main drawback of approaches using local weights is their computational complexity. Although local MSD weights based weighted voting strategy has been found to perform better than some global fusion approaches such as majority voting and STAPLE. We consider the majority voting as a global fusion approach since it has the same global weight 1 for an entire input segmentation. The main advantage of using a local weight based strategy becomes clear with a look at figure 2.10. A global fusion strategy cannot select the locally good regions within different inputs.

Figure 2.10: Example to show limitation of global candidate segmentation combination strategies. Atlas images 1 and 2 have been registered to target image. Image 1 is generally better registered, except for the upper arm of the star. However, global strategies cannot evaluate registration performance locally. Therefore, they cannot take advantage of this fact to obtain a better fused segmentation. Adopted from Artaechevarria et al. [2009]

Artaechevarria et al. [2009] calculated weights for each voxel by looking at the neighborhood values within a predefined square window. The weights are calculated using NCC, NMI and MSD in the neighborhood values of corresponding voxel from deformed moving image and fixed image. I. Isgum and van Ginneken [2009] calculated local weights by considering the absolute intensity difference between the corresponding voxels of the deformed moving image and the fixed image. Note that Artaechevarria et al. [2009] computed MSD by using intensity difference between all the voxel within a window unlike Isgum's approach of using intensity difference for each voxel separately. Although Isgum afterwards used a gaussian smoothing filter on the weights. Langerak et al. [2010] make a point that such weights, based on success of registration, are sensitive to noise in the image data. Also, image similarity does not correlate perfectly with registration accuracy or, more importantly, segmentation performance.

Problem with multi-atlas based segmentation fusion approach:

It is well proven that multi-atlas based segmentation fusion improves robustness against local registration failures Rohlfing et al. [2004a]. As long as the registration errors are non-symmetric and the atlases are independent (which is the case for inter-patient data), it is likely that the local errors in the various input segmentation can be corrected by combining these segmentations. However, there are two drawbacks, mentioned in Rohlfing et al. [2004a], on using all the available atlases: (1) The main drawback is the computational burden introduced by the multiple registrations and segmentation fusion ; (2) the shape variance in the observed set of atlases may not represent the unseen image.

2.5 Conclusion

In this chapter we have provided an overview of the literature relevant to the atlas based approach to segmentation. We employ the ABS approach for lobular segmentation of the pulmonary CT data. The intensity range of the lobes is similar to that of the fissure, therefore several popular intensity based segmentation approaches such as active shape models, level sets etc. cannot be used in this application. This makes the choice of ABS approach a logical option. A brief review of the image registration is provided since understanding registration is pivotal for understanding the ABS process. We saw that ABS is a two step process. In the first step, a moving image is registered to an unseen fixed image. This gives us a transform from the moving image domain to the fixed image domain. In the second step, the transform obtained in the first step in applied on the segmentation of the moving image. This gives us an estimate of the segmentation of the unseen fixed image. The dependency of final segmentation accuracy on the registration encourages to use multiple atlases. In literature several ways of combining information from many atlases are proposed: creating an average atlas, probabilistic atlas, choosing a single best atlas, or segmentation fusion based approach. We select the segmentation fusion based multi-atlas approach to research further since it has been proven to be more accurate than other approaches. In this approach, several atlases are independently registered to an unseen image. These independent ABSs give us several estimates of the segmentation of the unseen image, or several estimates of the ground truth. A fusion of these estimates can be more accurate than any individual estimate, if the registration error in the individual ABSs are non-symmetric, which is usually the case for inter-patient data.
The task of segmentation fusion is analogous to combining several pattern classifiers. In the context of multi-atlas based segmentation, an input segmentation can be seen as a decision from one classifier. Therefore, several estimates of ground truth segmentation can be viewed as classification decision from many pattern classifiers. This makes the segmentation fusion a typical problem of combining the pattern classifiers. Majority voting and the weighted voting are some of the popular approaches of combining pattern classifiers, which are frequently used for segmentation fusion also. In the current chapter we saw a review of such approaches with a detailed discussion on how the weighted voting has been proven to outperform the majority voting. Majority voting is undoubtedly the most popular way to combine the classifiers. It is most intuitive to agree with a decision with maximum support. It has been shown to give good results with very less computations involved. The problem with majority voting is that it considers all the inputs to be equally accurate while some of the inputs may have entirely wrong segmentation decision. Majority voting may be the right choice for fusion if all the inputs are equally important and there is no need to assign different weights to the input segmentations. But if input segmentations may vary largely from each other, it is a good idea to find the weight associated with inputs. In weighted voting, each segmentation is assigned a weight value which

shows the probability of this segmentation being correct. Once we decide to use weighted voting based fusion strategy, finding weights is the main task.

This chapter discusses various approaches to compute the weights for individual segmentations. We have divided all the weight calculation approaches in two broad categories: performance based weights and regional weights. In case of performance based weight calculation approaches, weights are calculated either by using a similarity measure between the deformed moving image and the fixed image (such as MSD, NCC, or NMI between the deformed moving and the fixed image) or by comparing input segmentations with an estimate of the ground truth (as is done in the case of STAPLE and SIMPLE algorithms). We studied the comparison between the setups of the STAPLE and the SIMPLE algorithm. SIMPLE has been reported to outperform STAPLE significantly. In case of regional weight calculation approaches, either a global weight is calculated for an input segmentation or a separate local weight is computed for each voxel of the segmentation. There are trade-offs among different approaches. The use3 of global weights is computationally less expensive, but it may give less accurate result as compared to local weight based segmentation fusion approaches. Of course, this would be the case if the input set has locally non-symmetric errors.

The input segmentations, created by multi-atlas based segmentations, for our pulmonary CT scan data are not equally accurate. Therefore a weight based fusion strategy has higher chances of outperforming the majority voting. Also, the input segmentations for our dataset are found to be locally good. This happens because of locally correct registration. This means that in some local region, registration may be more accurate than other regions. Therefore a local weight based fusion strategy seems to a good choice for the kind of inputs we are working with.

Chapter 3

Data, Experimental Setup and Registration Setup

3.1 Data and Experimental Setup

The data used in this experiment contains CT scans from 23 COPD patients. Manual segmentation and lung masks of the entire dataset are prepared by experts in the LKEB group. The left lung is segmented into 3 regions: upper lobe (intensity value 2), lower lobe (intensity value 1), and background (intensity value 0). Similarly, the segmentation of right lung contains superior lobe (intensity 3), middle lobe (intensity 2), lower lobe (intensity 1), and background (intensity 0).

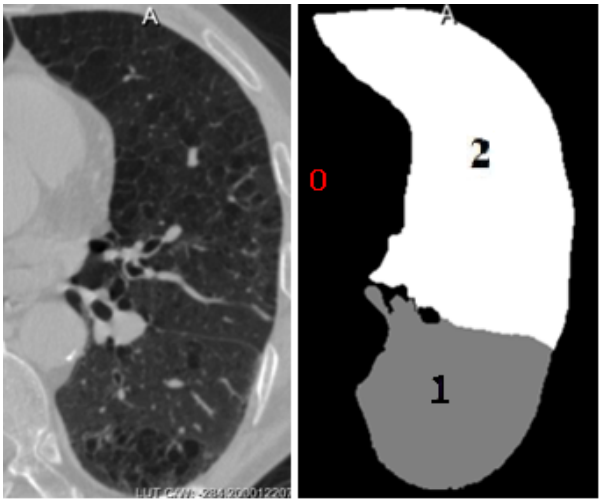


Figure 3.1: (Left) An example left lung CT scan; and (Right) its segmentation

On an average, the resolution of one lung is approximately 250 × 350× 600 pixels. In our experiment, we have processed both the lungs one by one. This was done mainly to improve computation speed of each algorithm. We conducted the experiment in a leave-one-out fashion. Therefore, each atlas is once selected as unseen image and all the rest atlases are transformed to it.

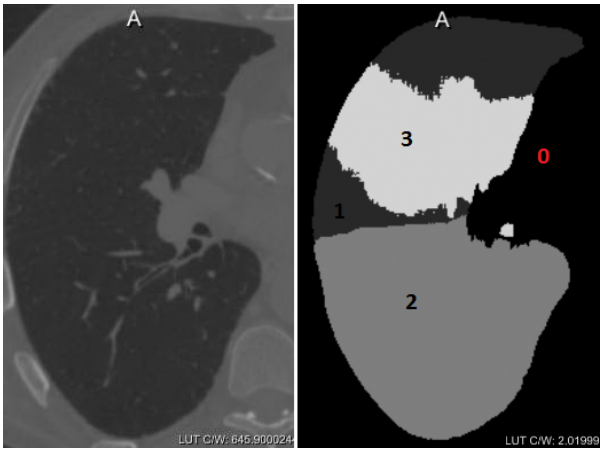


Figure 3.2: (Left) An example right lung CT scan; and (Right) its segmentation

3.2 Softwares Used

Coding is done in C++ using ITK. Python scripts are used to call the executables. We use $MeVisLab$, a medical image processing and visualization software, for visualizing images. MATLAB is used for making boxplot of the results and wilkoxon test presented in this report.

3.3 Image Registration

3.3.1 Elastix

All the registrations in the current work are performed using a registration package `elastix` developed by Staring et al. [2010]. Executables and source code of `elastix` are publicly available from the website `http://elastix.isi.uu.nl`. On the same site one can also find a database of 'parameter files', where each file contains a set of registration parameters, together with a short description of the clinical application for which they were used.

3.3.2 Image registration setup

The CT scan images are obtained from different patients and their resolutions are different as compared to each other. An affine transformation is not found to be sufficient and non-rigid registration is needed. Therefore, we employ a three step registration strategy (as was used in Staring et al. [2010] for *intra-patient* pulmonary CT data) in order to transform moving images M_i to the domain of fixed image F.

Effect of masking: As was mentioned in Staring et al. [2010], we make use of 'masked' fixed image and 'masked' moving images for nonrigid registrations in step-2 and step-3. All the voxels outside the lung segmentation are given the intensity value 0. This was done to mitigate the effect of the ribs on the transformation within the lungs. The image gradient is relatively high at the ribs which influences the deformation field, and causes misalignment of fissures and vessels. This masking step eliminates the problem caused by mismatching of the ribs. The usage of mask provides a better match of smaller structures within the lung.

1. Affine registration of each moving image M_i to the fixed image F independently, without using lung masks (table 3.1) for the transform parameters. We obtain transform T_{i1} as a result of this registration. The affine registration is provides a coarse global alignment.

Lung masks are not used in order to use the structures in the background part of the images. Their presence helps in a better global alignment.

2. Nonrigid registration of each *'masked'* moving image M_i to the *'masked'* fixed image F independently, without using lung masks (table 3.1) for the transform parameters. The transform T_{i1} obtained in step1 is used as initial transform for this registration. We obtain transform T_{i2} as a result of this registration. It was mentioned in Staring et al. [2010] that the use of lung masks at an early stage shows negative impact on lung boundary alignment.

3. Nonrigid registration of each *'masked'* moving image M_i to the *'masked'* fixed image F independently, with the use of lung mask of the fixed image (table 3.1). The transform T_{i2} obtained in step1 is used as initial transform for this registration. We obtain transform T_{i3} as a result of this registration. This transform T_{i3} is applied to the corresponding segmentation S_i, which gives deformed segmentation DS_i.

3.3.3 Problem with the current registration

The above mentioned registration setup has proven to work well for intra-patient pulmonary CT data, while we are dealing with inter-patient data. In former case, presence of vessels help with registration as they work as landmarks while it is not possible to match vessels for the data obtained from different patients. Since different people have quite different alignment of vessels which is bottleneck for obtaining good registration. In addition to mismatch of vessels, extremely thin fissures also suffer from registration. Lets have a look at one of registration result obtained from the 3-step approach discussed in the section 3.3.2. See figure 3.3.
As can be seen in the figure 3.3, the vessels and fissure are severely distorted inside the lung region. Although the lung boundaries in the deformed moving image DM_i match quite well with that of the fixed image. This severely distorted fissure in the deformed moving image causes Since our main focus is on the lobe segmentation, it is important to detect the fissures correctly. Therefore, it is of prime importance to 'fix' registration in order to retain the fissure structure in the deformed moving image.

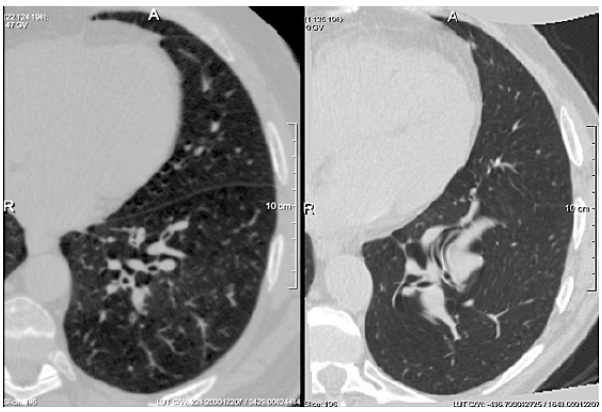


Figure 3.3: (Left) Fixed Image; (Right) One of the deformed moving images. Note that the region inside lung boundary in completely distorted in the deformed moving image. The fissure is clearly visible in the fixed image while it is completely lost in the deformed moving image.

The reason of such poor registration is two fold:

Table 3.1: Parameter settings for stages 1-3 in registration setup-A (section 3.3.4.1), resolution levels R1-R5. Adopted from Staring et al. [2010]

Stage	Iterations		R1	R2	R3	R4	R5
1. Affine	1000						
2. Nonrigid without mask	1000	Grid spacing (mm)	80	80	40	20	10
		Downsample factor	16	8	4	2	1
3. Nonrigid with mask	2000	Grid spacing (mm)	80	40	20	10	5
		Downsample factor	4	3	2	1	1

1. Presence of vessels causes problem in registration. In case of intra-patient data, vessels may help in registration. But in case of inter patient data, it is difficult to register the vessels from moving to the fixed image. Therefore we see results such as figure 3.3.

2. The fissures are very thin and their intensity range is same as that of the lobes. Therefore it is difficult for registration to differentiate the fissures from rest of the lobe region. Consequently, fissures are almost completely eroded in the registered atlas. See figure 3.3.

3.3.4 Rectifying registration

We experimented with two different registration strategies which give agreeable results for our dataset.

3.3.4.1 Registration setup A

In this setup, we enhance the fissures in unseen and atlas images , before registration. If intensity of fissure distinctly vary from that of the nearby lobe region then it may help in registration. We adopt following strategy for enhancing the fissures in both fixed and moving image, before registration. See the appendix 9 for details of the transform parameters used in the registrations.

1. Fissure or sheetness detection: A sheetness filter, developed in the LKEB/LUMC group, is used to detect the fissures in pulmonary CT data. The sheetness filter not only responds to fissures, but unfortunately also to vessels, since at smaller scale the outer vessel boundary resembles a sheet. See figure 3.4.

2. Vessel detection: In order to remove vessels from the response of sheetness filter , we detect the vessels using a vesselness measure described in Xiao et al. [2011]. See figure 3.4.

3. Removing vessel responses from sheetness filter response: The vessels detected in the second step are subtracted from the response of sheetness filter from the first step. Although vesselness measure does not detect all the vessels that are present in the response of sheetness filter. Therefore, the resultant image obtained after vessel subtraction still contains some vessels, in addition to fissure.

4. Connected component analysis: We retain one largest connected component from the image obtained form vessel subtraction. The resultant image contains only fissure as shown in 3.4 and 3.5.

5. Enhancing the fissure: Once we have an initial estimate of fissures, we scale down the intensity of all voxels in the fixed and the moving image by a constant factor, except at the fissure location. This way we suppress the vessels so that they do not distract registration. We also mask the background from the images in order to remove ribs for the same reason as mentioned in section **??**. Let us call the resultant image from this step as *enhanced fixed image (EF)* and i^{th} *enhanced moving image* (EM_i).

The first 4 of the overall 5 steps discussed above are for obtaining an initial estimate of fissure in order to support registration. These 4 steps are used in a previous research conducted in LKEB. The registration setup A is a 3-step strategy (1 affine and 2 b-spline registrations) in which first 2 steps are same as section 3.3.2. In the third step, we perform nonrigid registration of each enhanced moving image EM_i to the enhanced fixed image EF independently. The transform T_{i2} obtained in step1 is used as initial transform for this registration. We obtain transform T_{i3} as a result of this registration. This transform is applied to the corresponding segmentation S_i to obtained deformed segmentation DS_i.

3.3.4.2 Registration setup B

van Rikxoort et al. [2010a] proposed a registration strategy for pulmonary CT scan data in which an initial estimate of fissure, bronchial-tree and lung border is used to improve registration quality. Registration setup-B, proposed by Dr. Marius Staring, is a modified version of their method. We do not compute bronchial-tree. But we compute an initial estimate of fissure using the work of Xiao et al. [2011], similar to what we saw in in setup-A. Setup-B is a 2-step registration strategy (1 affine and 1 b-spline registration) contrary to 3-steps (1 affine and 2 b-spline registrations) involved in setup-A. Following are the steps involved in the setup-B:

1. Affine registration of each i^{th} *masked* moving image to the *masked* fixed image independently. We obtain transform T_{i1} as a result of this registration.
2. Compute lung border for each moving image M_i and the fixed image F.
3. Make an initial estimate of fissure for each moving image M_i and the fixed image F. For this we the employ approach discussed in first 4 steps of setup-A.
4. Compute distance-map of lung border of each moving image $DT(M_{i,border})$ and the fixed image $DT(F_{border})$. Both the distance maps are truncated after a certain threshold (value 14 used in our experiment).
5. Compute distance map of initial estimations of fissure of each moving image $DT(M_{i,fissure})$ and the fixed image $DT(F_{fissure})$.
6. Non-rigid registration using cost function as given in equation 3.1. The transform T_{i1} obtained in step1 is used as initial transform for this registration. We obtain transform T_{i2} as a result of this registration. This transform T_{i2} is applied to the corresponding segmentation S_i to obtained deformed segmentation DS_i.

$$\begin{aligned}\hat{\boldsymbol{T}_{\boldsymbol{\mu}}} = \arg\min_{\boldsymbol{T_\mu}} \alpha_1 \mathcal{C}_1(\boldsymbol{T_\mu}; DT(F_{border}), DT(M_{i,border})) \\ + \alpha_2 \mathcal{C}_2(\boldsymbol{T_\mu}; DT(F_{fissure}), DT(M_{i,fissure})),\end{aligned} \tag{3.1}$$

See the appendix 9 for details of the transform parameter used in the registrations and the parameters α_1 and α_2 in the equation 3.1.

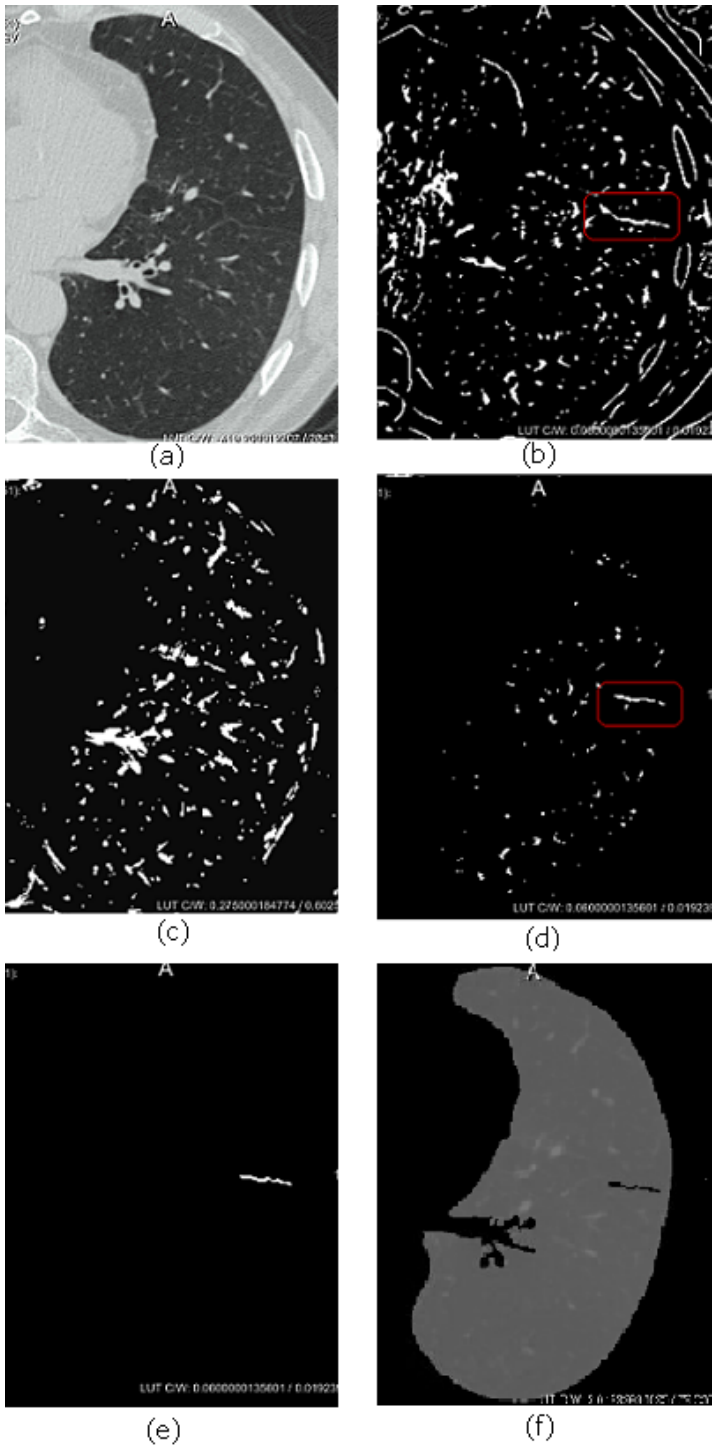


Figure 3.4: (a) A CT scan image of left lung, (b) Response of sheetness filter. We can see the encircled fissure and lots of other vessels, (c) Response of vesselness filter. It does not detect all the vessels that are present in the response of sheetenss filter, (d) Image obtained on subtracting (b) - (c). We can see that some of the vessels are still present in addition to fissure, (e) Fissure obtained after connected component analysis of image from step (d), (f) The *enhanced* CT image.

Figure 3.5: 3D view of the fissure obtained from connected component analysis. (a)Axial view, (b)Sagitaal view, (c)Coronal view.

3.4 Result Format

3.4.1 Overlap of fissures

Accuracy of a resultant segmentation obtained from any fusion algorithm is calculated by considering its overlap with the ground truth. Since focus of the current work is lobular segmentation, a resultant segmentation is considered more accurate if its fissure has higher overlap with that of the ground truth. As can be seen in figure 3.6, the overlap value of entire 3D segmentation result with the ground truth is usually high and it does not convey much information about how good the detected fissure is. Therefore, accuracy of a segmentation result is reported in terms of the overlap of its 3 voxel wide fissure with that of the ground truth as shown in figure 3.7.

Figure 3.6: (a1, a2) Two input images (b) Ground truth; The number shows overlap value of corresponding input images with the ground truth.

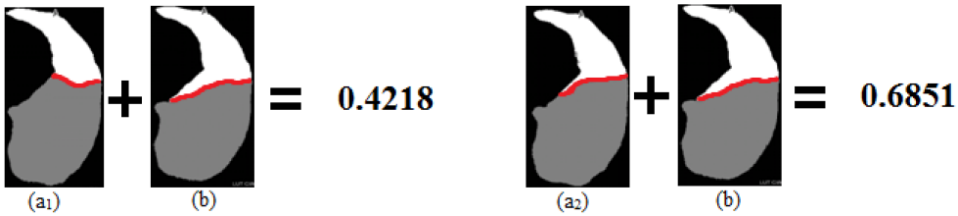


Figure 3.7: (a1, a2) Two input images (b) Ground truth; The number shows overlap value of 3 voxel wide fissures from the ground truth with that of the input images.

3.4.2 Box-plot of fissure overlap

The presented work focuses on multi-atlas segmentation based label fusion strategies (section 2.4) for obtaining the lobular segmentation. The experiments are conducted in a leave-one out fashion. Therefore, each of the 23 input CT images is taken as fixed image once and remaining CT images are registered to it, which eventually results into an estimate of segmentation of the fixed image. Finally, we obtain 23 estimates of segmentation of the fixed image as a result of 23 independent atlas based segmentations. Therefore, we obtain 23 estimates of fissure in the ground truth. The results are reported by making a box-plot of the overlap of these 23 fissures with that of the ground truth.

3.4.3 Box-Plot of fissure overlap for majority voting

On using majority voting as the label fusion strategy, we obtain a box-plot as shown in figure 3.8

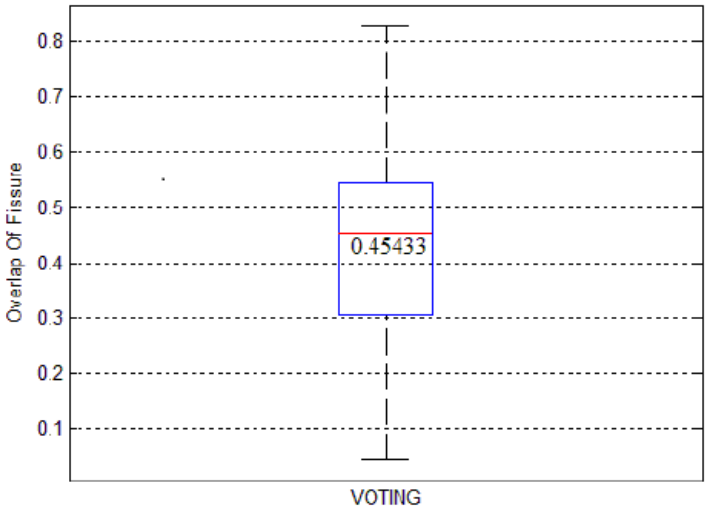


Figure 3.8: Box-plot of the 23 overlap values obtained from estimates of ground truth fissure. These estimates are obtained by using majority voting as the label fusion strategy.

3.4.4 Low tail of box-plot

The fissures may not always be visible in standard CT images. In such a situation, initial fissure estimation may not be able to detect anything, which further causes poor registration. Therefore, we may encounter a set of deformed segmentations DS_i as shown in figure 3.9. No combination strategy can produce a good estimate of ground truth on using the inputs shown in the figure because none of the inputs contain *locally* correct fissure. Such cases result into either 0 or very less fissure overlap on comparing with the corresponding ground truth. This causes the tail of box-plot to be quite low (figure 3.8). It should be noted that one needs to improve registration for improving this result and not the segmentation fusion strategy.

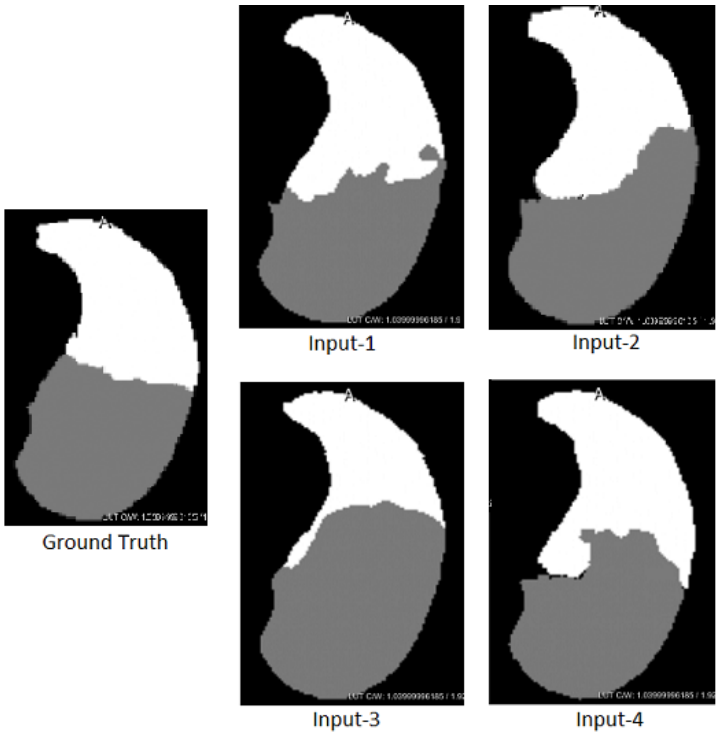


Figure 3.9: Reason for low tail of box-plot: A set of four input segmentations and their corresponding ground truth is shown. Such inputs are generated due to failure of registration process. Since none of the input is locally correct, one cannot expect good result from any fusion strategy.

Chapter 4

Improvements to SIMPLE: Performance Based on Estimate of Ground Truth

Several researchers have used majority voting for decision fusion in the context of multi-atlas based segmentation. Majority voting has established itself as a powerful tool for decision fusion. We can simply trust the decision of majority voting for the region where clear majority is present. By clear majority we mean the situation when one of the labels has quite high support as compared to others. If we look at our pulmonary CT data, all the 22 input segmentations (for any given fixed image) have same label for most of the region. But inputs differ from each other (figure 4.1) in the region close to fissure. Therefore, using majority voting works good for entire image but the region nearby fissure.

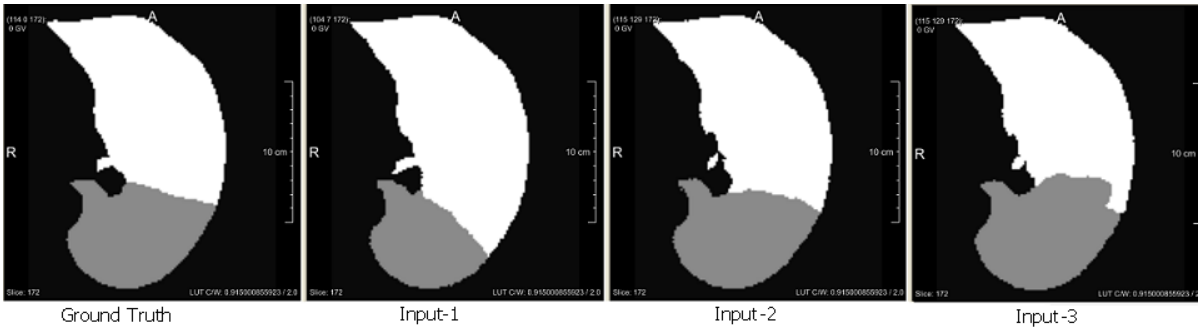


Figure 4.1: For most of the region, all the 3 inputs have same label as that of the ground truth , therefore a clear majority is present. But for region close to fissure, input-2 looks more similar to ground truth compared to others. Therefore the fusion strategy should assign higher weight to input-2 as compared to other two inputs. But majority voting assigns default 1 weight to each input.

Majority voting assumes equal weight, i.e. 1, for each input segmentation. If an input segmentation may be more accurate than others, it should be given higher *weight*. Therefore, one should consider *weighted voting* instead of majority voting for fusing such inputs. There are several methods for finding weight of an input segmentation. In this chapter we have discussed several algorithms which compare input segmentations with an estimate of ground truth for finding weights. This chapter is focussed on one such algorithm, SIMPLE. SIMPLE is an itera-

tive algorithm. It calculates a weight value for each input segmentation and performs weighted voting of all the segmentations. This task of weight calculation, then weighted voting, is repeated in each iteration in such a way that weights and the result of weighted voting improves over iterations until the process converges. The result of weighted voting in the final iteration is considered as the final estimated ground truth. In each iteration, SIMPLE computes DICE overlap of input segmentations and current estimate of ground truth. These DICE values are used as weights for performing weighted voting.
We have experimented with several improvements over SIMPLE in which we used the same DICE measure as weight for input segmentations, but we focus on improving the weights in a way that they are more representative of *locally* good regions in a segmentation. Also, SIMPLE algorithm works only for binary segmentation. We propose a *multi-label* version of SIMPLE.
This chapter contains results generated only for left lung. All the input segmentations were obtained through multi-atlas based segmentation and setup-A (see section 3.3.4.1 for details) was used for performing all the registrations. Section 4.1 contains detail, pseudo code, and an example of SIMPLE. Various improvements over SIMPLE are explained in section 4.2 in detail. This section also discusses the motivation of proposed algorithms. In section 4.3 we have presented the results of various algorithms and their discussion. Section 4.4 concludes.

4.1 SIMPLE Algorithm

SIMPLE uses an iterative strategy which starts with computing an initial estimate of ground truth by majority voting of all the input segmentations. Furthermore, DICE overlap of each input and this initial estimate is calculated. Inputs with DICE value less that a certain threshold are rejected i.e. their weight is made 0. This threshold is computed based on average and standard deviation of the DICE values of inputs. The DICE values are used as weights in weighted voting of the remaining inputs. The result of weighted voting provides us with a new estimate of ground truth. Again, overlap of each remaining input is computed with this new estimate. Therefore, the process enter into a loop and in each iteration some of the inputs are rejected while the process continues with the remaining ones. The process stops when inputs are not rejected anymore. See algorithm-1. Section 4.1.1 demonstrates working of SIMPLE through an example.
One important note is mentioned in Langerak et al. [2010] that SIMPLE method may fail occasionally, i.e. lead to a clinically unacceptable segmentation. This happens when the initial combination of input segmentations $DS_i (i = 1, \cdots, N)$ does not resemble the unknown ground truth segmentation. This will mainly be the case when most of the atlas based segmentations fail because of poor registration. The solution to this problems lies in improving registrations, rather than improving segmentation fusion scheme.

4.1.1 An example of SIMPLE

Consider an example given below: Lets assume 6 inputs with overlap values (0.95, 0.97, 0.92, 0.89, 0.60, 0.75) with initial estimate of ground truth. Consider $\alpha = 0$ (see step-9 in algorithm-1). In the first iteration $Threshold_0 = Mean_0 = 0.85$. Therefore inputs with 0.60 and 0.75 overlaps are rejected and estimate of the ground truth is updated by performing weighted voting of remaining 4 inputs. Now, the overlap values of the inputs change on comparing with the updated estimate of ground truth. Lets assume it to be (0.94, 0.99, 0.94, 0.86). In second iteration, $Threshold_1 = Mean_1 = 0.93$. Therefore input with 0.86 overlap is rejected and the

Algorithm 1 SIMPLE algorithm

Require: DS_i, α

perform majority voting of DS_i to obtain an initial estimate of ground truth, i.e., $E(GT)_0$
initialize $iter = 0$
initialize $NumberOfInputs_0 = N$
while $NumberOfInputs_{iter} \neq NumberOfInputs_{iter-1}$ **do**
  for $i = 1$ to $NumberOfInputs_{iter}$ **do**
    compute $Dice_i$ = overlap of DS_i and $E(GT)_{iter}$
  end for
  compute $Mean_{iter}(Dice)$ and $StdDev_{iter}(Dice)$
  compute $Threshold_{iter} = Mean_{iter} - \alpha \times StdDev_{iter}$
  for $i = 1$ to N **do**
    if $Dice_i < Threshold_{iter}$ **then**
      reject DS_i
    end if
  end for
  count $NumberOfRejectedInputs_{iter}$
  $NumberOfInputs_{iter+1} = NumberOfInputs_{iter} - NumberOfRejectedInputs_{iter}$
  $iter = iter + 1$
  perform weighted voting of remaining DS_i to obtain $E(GT)_{iter}$. $DICE_i$ is used as weight for DS_i.
end while
$E(GT)_{iter}$ is final fusion result, i.e., $E(GT)$ (estimated ground truth)

process continues until we stop rejecting inputs. This example gives an idea of how rejection of data helps in improving the 'threshold' over iterations.

4.1.2 Binary input for SIMPLE

Since SIMPLE is designed to work for binary segmentation only and segmentation of a left lung contains three labels in our data, we chose to make the label of upper lobe same as that of the background i.e. zero, figure 4.2. We can still locate fissure using the binary inputs.

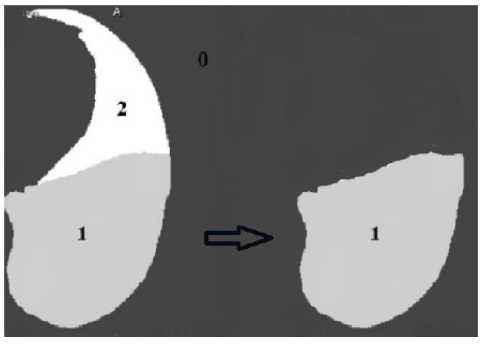


Figure 4.2: (Left) Segmentation image for a left lung, (Right) Binary input to binary SIMPLE algorithms.

4.2 Improvements in SIMPLE Algorithm

SIMPLE is proven to attain significantly higher accuracy as compared to majority voting Langerak et al. [2010]. But there are possible improvements which may help SIMPLE in performing even better. In this section we have proposed several modifications in SIMPLE and their theoretical contribution is discussed in detail.

4.2.1 Local SIMPLE

In SIMPLE, the performance of an input segmentation is represented by a global weight value which is obtained by computing its overlap with current estimate of ground truth. Therefore, all the local regions of an input segmentation are assumed to be equally accurate and each voxel is weighted by the same global overlap value. We have already discussed in section 2.4.4 that a combination strategy based on local weights is expected to perform better as compared to one based on global weights. Since an input segmentation may be locally good in some area while bad in some other area. Let's have a look at figure 4.3 in which some of the input segmentations and their corresponding ground truth is shown. Since our focus is on fissure detection only, fissures in the input segmentations are highlighted if they match with the fissure in the ground truth. As can be seen, none of the input segmentation has completely correct fissure while most of the inputs have local regions in which their fissure matches well with that of the ground truth. Therefore it is reasonable to assign different weights to different local regions within the same input segmentation. We have proposed 3 different methods which employ *local weights*:

1. Local slice-wise
2. Local non-overlapping cube-wise
3. Local overlapping cube-wise

4.2.1.1 Local slice-wise SIMPLE

As was mentioned in section 3.1, the input segmentations are $3D$ images. An input image can be considered as a number of 2D slices stacked over each other along a particular dimension. The algorithms starts with an initial estimate of ground truth by majority voting of input segmentations. Furthermore, DICE overlap of each slice of an input is calculated w.r.t corresponding slice of the initial estimate of ground truth. In this setup, we get a separate mean overlap and threshold for each slice index. Therefore, instead of rejecting entire $3D$ input segmentation, particular slices are rejected if their overlap is less than the threshold for that slice index. In this methodology, we focus on rejecting only *bad* slices from an input and remaining good ones can still participate in weighted voting. Similar to SIMPLE, it is an iterative procedure and the process continues until no slice is rejected from any input. See algorithm-3 for pseudo code of local slice-wise SIMPLE given in the appendix 9.3.

4.2.1.2 Local non-overlapping cube-wise SIMPLE

This algorithms is quite similar to 'local slice-wise' SIMPLE since we focus on finding weights for local regions separately. Each local region is assigned a separate threshold to compare with. Some of the local regions may be rejected from an input, but the remaining ones still participate

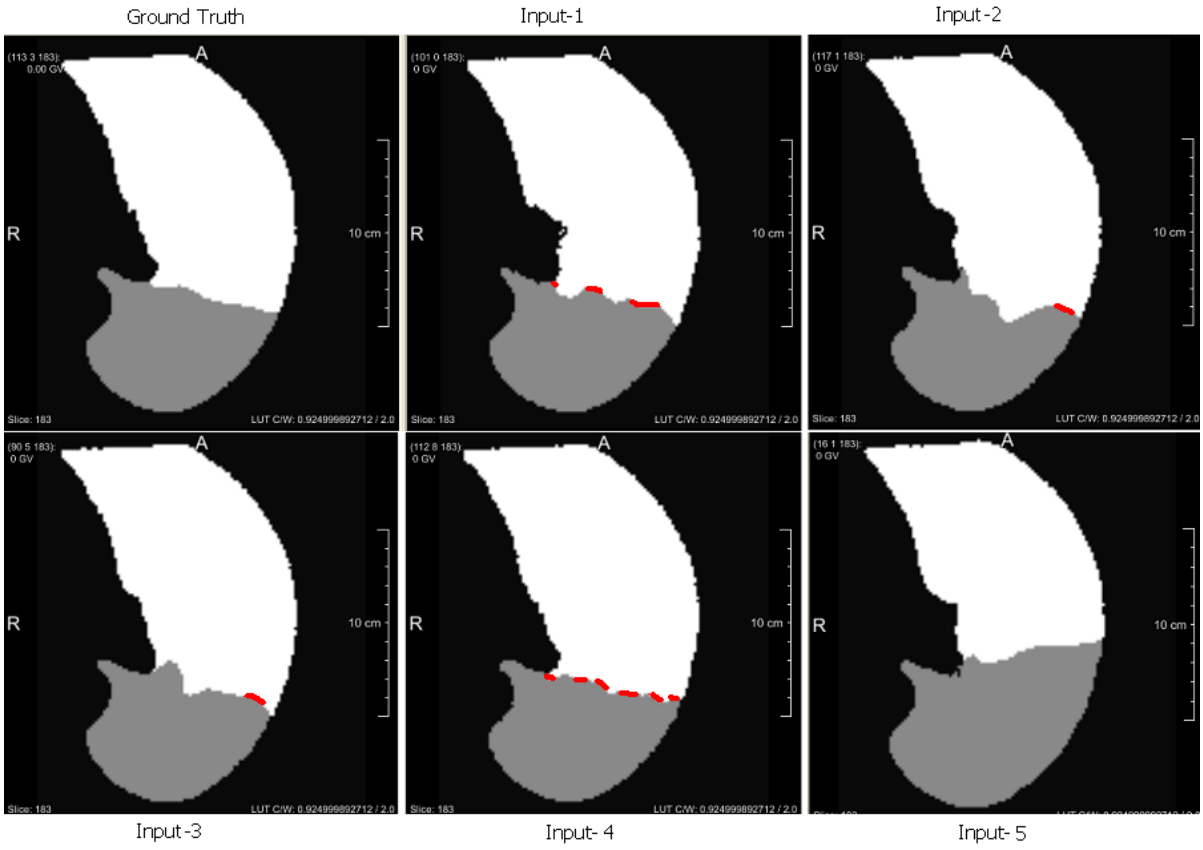


Figure 4.3: For most of the region, all the 3 inputs have same label as that of the ground truth , therefore a clear majority is present. But for region close to fissure, input-2 looks more similar to ground truth compared to others. Therefore input-2 should be assigned more weight. While using majority voting here assigns 1 weight to each input.

in weighted voting. This is also an iterative process which continues unless we stop rejecting any local region from any input. The main difference from 'local slice-wise' algorithm is the division of an input image into small non-overlapping cubes of pre-defined size. Also, this algorithm need the cube-size as user input. See algorithm-4 for pseudo code of the local non-overlapping cube-wise algorithm given in the appendix 9.3.
If we look at figure 4.3, fissure in input-5 does not match with the shown ground truth at all. Considering this ground truth to be the current estimate, all the cubes containing fissure in input-5 have high chances of being rejected. But remaining cubes with label-1 may still participate.

4.2.1.3 Local overlapping cube-wise SIMPLE

In 'local non-overlapping cube-wise' algorithm an input image is divided into smaller non-overlapping cubes. Instead we can divide an image into overlapping cubes. Weight for each voxel of an input image is calculated by considering the overlap of its 3D neighborhood with the neighborhood of corresponding voxel from estimate of ground truth. Therefore, each voxel from an input segmentation is rejected or retained based on the overlap value of its neighborhood. This algorithm also needs a user to input the cube-size. The effect of changing the size of overlapping cubes is governed by two effects:

1. The subdivision into overlapping cubes has 'averaging effect' on local weight values. It ensures smooth variation of weights. For instance, assuming a $2D$ image and a square window of 10×10 pixels, on shifting the square window by 1 unit (along one of sides of square), two neighboring pixels share 90 out of 100 pixels which are used to compute local weight. This large number of shared pixels provides smooth variation of weights along an image. On increasing the neighborhood radius, the averaging effect also increases. But such weights are less local and may result into worse performance. Therefore, it is expected that on increasing the size of the sliding window (or cube size), the performance should improve initially because of reducing effect of noise and worsen further because of increasing global nature of weights.

2. It is possible that a very small window does not contain enough information to calculate local weights reliably. Therefore, smaller window size (or cube size) may result in poor performance, while increasing window size may improve the performance.

It is difficult to predict which one of the above two effects may dominate, for a given cube size.
Improving Computation Speed for the Algorithm: Local overlapping cube-wise algorithm calculates DICE overlap value for each voxel of an input image. This value is computed by considering all the voxels, within a $3D$ window of given size, in the neighborhood of the voxel in consideration. Afterwards, this window is shifted by just one voxel along a particular dimension. It incurs a lot of computation which can be avoided, since majority of the voxels are still same within a window, on shifting it by just one voxel. Therefore, we shift the window by more than one voxel space,e.g., consider local overlapping cubes of size $50 \times 50 \times 50$ voxels. If we shift the window by 5 voxel size, instead of 1 voxel, then it gives us approximately $5 \times 5 \times 5 = 125$ times higher computation speed. Since DICE value is calculated for 125 times lesser number of cubes. Now, we have two parameters in this algorithm: the window size (also termed as cube-size in this report) and the window shift.

4.2.2 Increasing sensitivity (boundary) SIMPLE

SIMPLE algorithm computes accuracy of an input segmentation by its overall overlap with the current estimate of ground truth. This overall overlap value may not be a good representative of how good an input segmentation is. Since even with poor boundaries, a greater part of an input segmentation still matches with current estimate of ground truth. Therefore the overlap value is always quite high. We can obtain a more sensitive measure of goodness of an input segmentation by computing the overlap of its boundaries with that of the current ground truth. In SIMPLE, the performance of an input is represented by overlap value of 3D input image with the current estimate of ground truth, which is also a 3D image. This overlap value of entire 3D volume may not be a good representative of boundaries of an input image, as shown in figure 4.4.
One can see in figure 4.4 that boundaries of second input are more similar to that of the ground truth but the overlap value of both the inputs are quite close to each other since most of the 3D volume of both the inputs overlap with the ground truth.
Instead of computing overlap of entire input volume, we consider overlap of only boundary of an input with the boundary of ground truth, as shown in figure 4.5. This may provide more clear distinction among the good and bad inputs. See algorithm-5 for pseudo code of the boundary-SIMPLE algorithm given in the appendix 9.3.

Figure 4.4: (a1, a2) Two input images (b) an estimate of ground truth; The number shows overlap value of corresponding input images with that of the ground truth estimate.

Figure 4.5: (a1, a2) Two input images (b) an estimate of ground truth; The number shows overlap value of 3 voxel wide fissures from the ground truth with that of the input images.

4.2.3 Re-inclusion of rejected data

In SIMPLE, input data is rejected on comparing with current estimate of ground truth. This ground truth improves iteratively. Once thrown, an input never participates back into fusion. One problem with this set up is that input data is thrown based on a comparison with ground truth which is still improving.Therefore, it is possible that a useful input is wrongly rejected in an initial iteration.
Consider an example given below: Lets assume 6 inputs with overlap values (0.95, 0.97, 0.92, 0.89, 0.60, 0.75) with initial estimate of ground truth. Consider $\alpha = 0$. In the first iteration $Threshold_0 = Median_0 = 0.91$. Therefore inputs with overlaps 0.89, 0.60 and 0.75 are rejected and estimate of ground truth is updated by performing weighted voting of remaining 3 inputs. Now the overlap values of inputs change when compared with updated estimate of ground truth. Lets assume it to be (0.94, 0.95, 0.95, 0.91, 0.65, 0.79). In second iteration, $Threshold_1 = Median_1 = 0.93$. Therefore inputs with overlaps 0.91, 0.65 and 0.79 are rejected and the process continues until we stop rejecting inputs. This example gives an idea of how rejection of the data helps in improving the 'threshold' over iterations. The estimate of the ground truth is updated by making use of only non-rejected inputs. The use of 'median' instead of 'mean' ensures that threshold is not affected by outlier overlap values.

4.2.4 Multi-label SIMPLE

SIMPLE algorithm is designed to work only for binary segmentation. It computes a global overlap value for foreground and this overlap value is taken as weight for all foreground voxels. The similar approach can be employed for a multi-label segmentation problem. If input segmentations have more than one label, overlap of each label can be computed separately by comparing with corresponding label present in the current estimate of ground truth.It should be noted that we need a separate threshold for each label. In this setup, only voxels with a particular label are discarded if their overlap value is less than a certain threshold. Therefore, it is possible to reject a set of voxels with a particular label while voxels with another label are retained within the same input image.

Algorithm 2 Re-Inclusion of Rejected Data

Require: DS_i, α

perform majority voting of DS_i to obtain an initial estimate of ground truth, i.e., $E(GT)_0$
initialize $NumberOfInputs_0 = NumberOfInputs$
initialize $iter = 0$
while $NumberOfInputs_{iter} \neq NumberOfInputs_{iter-1}$ **do**
  for $i = 0$ to N **do**
    compute $Dice_i$ = overlap of DS_i and $E(GT)_{iter}$ for all the input segmentations.
  end for
  compute $Median_{iter}(Dice_i)$ and $StdDev_{iter}(Dice_i)$ using all the input segmentations.
  compute $Threshold_{iter} = Median_{iter} - \alpha \times StdDev_{iter}$
  for $i = 0$ to N **do**
    if $Dice_i < Threshold_{iter}$ **then**
      reject DS_i
    end if
  end for
  count $NumberOfRejectedInputs_{iter}$
  $NumberOfInputs_{iter+1} = NumberOfInputs_{iter} - NumberOfRejectedInputs_{iter}$
  $iter = iter + 1$
  perform weighted voting of remaining DS_i to obtain $E(GT)_{iter}$. $DICE_i$ is used as weight for DS_i.
end while
$E(GT)_{iter}$ is final fusion result, i.e., $E(GT)$

4.2.4.1 Global Multi-Label SIMPLE

Global multi-label SIMPLE works in the same way as SIMPLE. A separate overlap value, mean, and threshold is computed for each label. The DICE overlap computed for a particular label is a global weight value, since all the voxels with this label are assigned the same DICE value as weight. In a way, this algorithm performs SIMPLE for each label independently but weighted voting considers all the labels simultaneously, at the end of each iteration. See algorithm-6 for pseudo code of the boundary-SIMPLE algorithm given in the appendix 9.3.

4.2.4.2 Local Non-Overlapping (and Overlapping) Cube-Wise Multi-Label SIMPLE

Local non-overlapping (or overlapping) multi-label SIMPLE algorithms works almost same as 'local non-overlapping (or overlapping) cube-wise SIMPLE' algorithm 4.2.1.2. But instead of rejecting (or retaining) all the voxels within a cube, only voxels with a common label are rejected (or accepted). If input segmentations have 3 labels then each non-overlapping cube in an input has 3 weight values associated with it. Each of these weight value corresponds to a different label. An input image is divided into non-overlapping (or overlapping) cubes depending on the choice of algorithm.

4.3 Results And Discussion

4.3.1 Effect of α

SIMPLE computes a unique threshold value in each iteration based on the mean and standard deviation of the overlap values of all the inputs. The equation for threshold calculation is as following:

$$Threshold_{iter} = Mean_{iter} - \alpha \times Stddev_{iter} \tag{4.1}$$

where $Mean_{iter}$ and $Stddev_{iter}$ represent the mean and standard deviation of overlap values of all the input segmentations for a particular iteration $iter$. This threshold also depends on a scalar variable α. In figure 4.6, α is given as a function of the number of selected segmentations as reported in Langerak et al. [2010]. No significant difference was found in final fusion result on using different α values. Although it was reported that this holds for cases in which there is a large set of atlases. In other words, although SIMPLE is not sensitive to the threshold rule, i.e. α, but this may be different for other (smaller) atlas sets. It should be noted that the number of inputs used Langerak et al. [2010] is 100 which is much larger than out dataset having only 22 atlases and an unseen image.

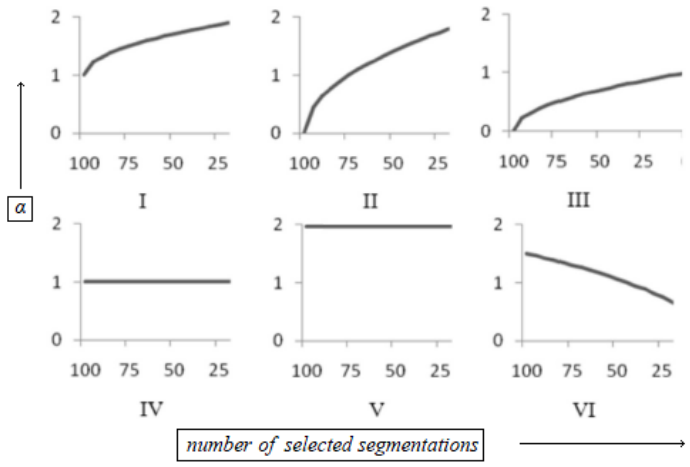


Figure 4.6: Selection threshold α (vertical axis) as a function of the number of selected segmentations (horizontal axis). Therefore α value in an iteration depends on the number of input segmentations that were selected in the previous iteration. Adopted from Langerak et al. [2010].

Effect of α on Our Data: The convergence speed of SIMPLE depends on the parameter α. It controls the amount of input data rejected in an iteration. The algorithm converges when inputs are not rejected any more. For a given α, SIMPLE may converge in 7 iterations with 30% inputs rejected while for another α, algorithm may converge in 5 iterations with 50% inputs rejected. Figure 4.7 shows performance of SIMPLE on our pulmonary CT dataset. The performance is represented by a boxplot of overlap values of 3-voxel wide fissure obtained from the SIMPLE result and the ground truth. See section 3.4 for the details on format of our result reporting. We experimented with several α values which are listed in table-4.1. f-value in the table denotes fraction of selected segmentations in previous iteration. We found statistically significant difference between the overlap values on using different α, e.g., Wilkoxon test statistic between the results obtained on using $\alpha = 0$ and $\alpha = 2$ is found to be 0.0021. Therefore, it is

important to choose the right α to get optimum performance from the algorithm. The α values given in table-4.1 are the ones for which SIMPLE worked best.

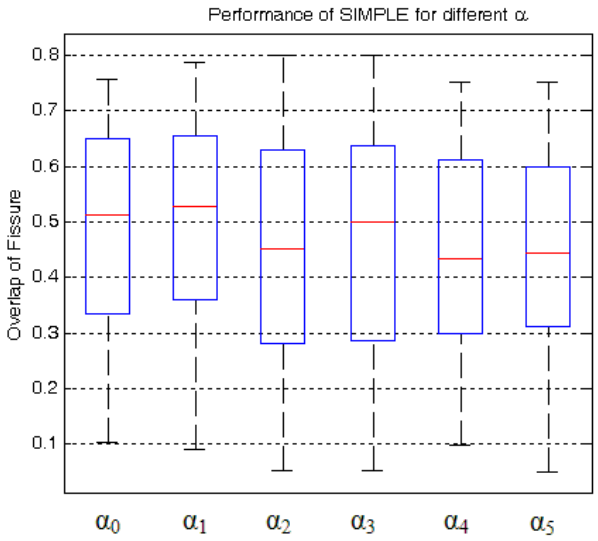


Figure 4.7: Box-plot of the 23 overlap values obtained on comparing fissures (3 voxel wide) obtained from ground truth and SIMPLE algorithm for different values of α.

Table 4.1: *Various α values used for SIMPLE algorithm.*

Index	α-value
0	$\sqrt{(1-f)}$
1	0.0
2	0.5
3	1.0
4	1.5
5	$1/f$

The convergence criteria of SIMPLE does not consider performance change over iterations, therefore it is possible that the accuracy of fusion result keeps on decreasing, but the algorithm still does not converge until α allows to reject more input data. Therefore, the algorithm does not keep any track of whether it is improving the final segmentation or making it worse. Therefore role of α becomes more important to ensure that the algorithm stops before it starts deteriorating the performance.

α Tuning Specific To Each Algorithm: We experimented with several α values for each of the proposed modified SIMPLE algorithm. A set of six α values is chosen separately for each algorithm. The results are calculated for each of these six α , as shown in figure 4.7, and one with highest value of median (α_1 in figure 4.7) is considered as the best result. The report contains only best result for each algorithm. It should be noted that different versions of SIMPLE algorithm may work best for different α. For these α values (given in table-9.1 and table-9.3 in Appendix), an algorithm's performance has no statistically significant difference. Therefore, we may chose any of these α to test on an unseen image.

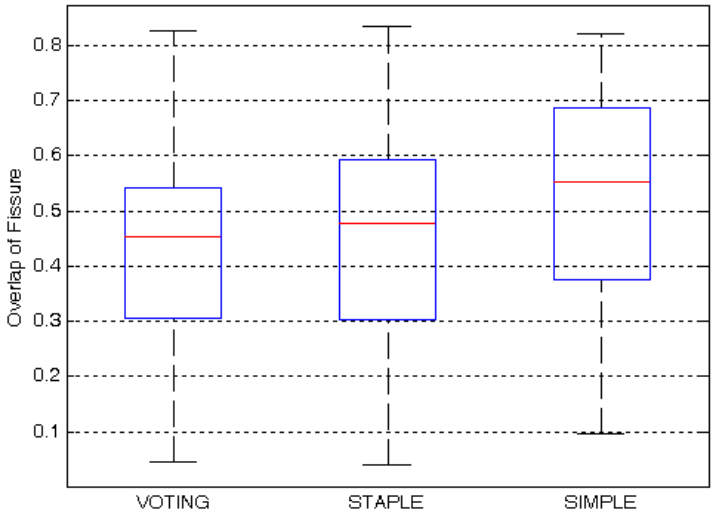
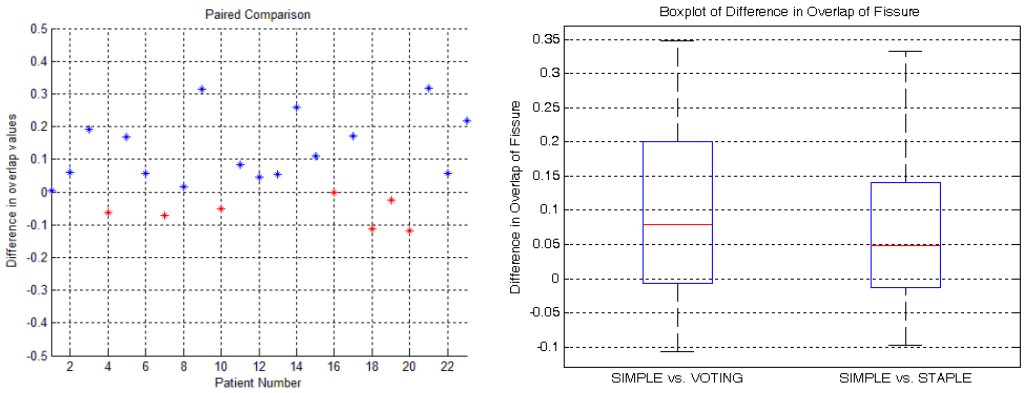


Figure 4.8: Box-plot of the 23 overlap values obtained on comparing fissures in label fusion result and ground truth. The fusion strategies used are majority voting (left) and SIMPLE (right).

4.3.2 Comparison with majority voting and STAPLE

Figure 4.8 shows performance of majority voting, STAPLE, and SIMPLE algorithms. It is evident that SIMPLE improves over majority voting and STAPLE. We analyze the difference of the corresponding overlap values obtained from majority voting and SIMPLE for detailed analysis. On doing a paired comparison for each patient (figure 4.9(a)), we see that SIMPLE improves fissure overlap for most of the patients but for some cases it performs worse, compared to majority voting. A box-plot of the difference of the corresponding overlap values can be seen in figure 4.9(b). Portion of the box-plot below 0 represents those cases where SIMPLE deteriorate the fissure overlap value compared to majority voting and STAPLE.



(a) The patient numbers for which SIMPLE improves over majority voting are shown in blue color while red marks show the patients for which SIMPLE performed worse than majority voting.

(b) The portion of the box-plot below 0 represents the cases when SIMPLE performed worse as compared to majority voting and STAPLE.

Figure 4.9: Comparing SIMPLE with majority voting and STAPLE. A plot of difference in overlap values of fissure obtained from various algorithms.

4.3.3 Results of proposed modifications in SIMPLE

Figure 4.10 shows box-plot of overlap of fissure obtained from SIMPLE and few other algorithms. Figure 4.11 contains box-plot of the difference in the overlap of fissure, obtained on subtracting fissure overlap value from various algorithms with that of the SIMPLE. We have discussed the result of each algorithm one by one.

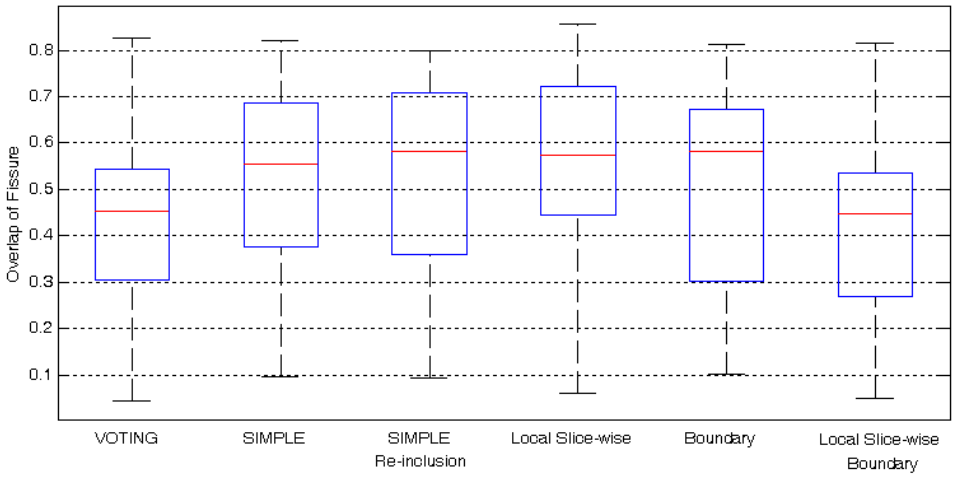


Figure 4.10: Box-plot of the 23 overlap values obtained on comparing fissures obtained from ground truth and various SIMPLE algorithms.

SIMPLE with re-inclusion algorithm: In figure 4.10, box-plot for 'SIMPLE with re-inclusion' algorithm shows a bit higher median compared to SIMPLE. Major part of box-plot for 'SIMPLE with re-inclusion' algorithm is above 0 in figure 4.11, which represents the examples where final fissure overlap was improved compared to SIMPLE. But we also see 2 outliers below 0-line, the examples for which SIMPLE performed better. We further do 'Wilkoxon-test'

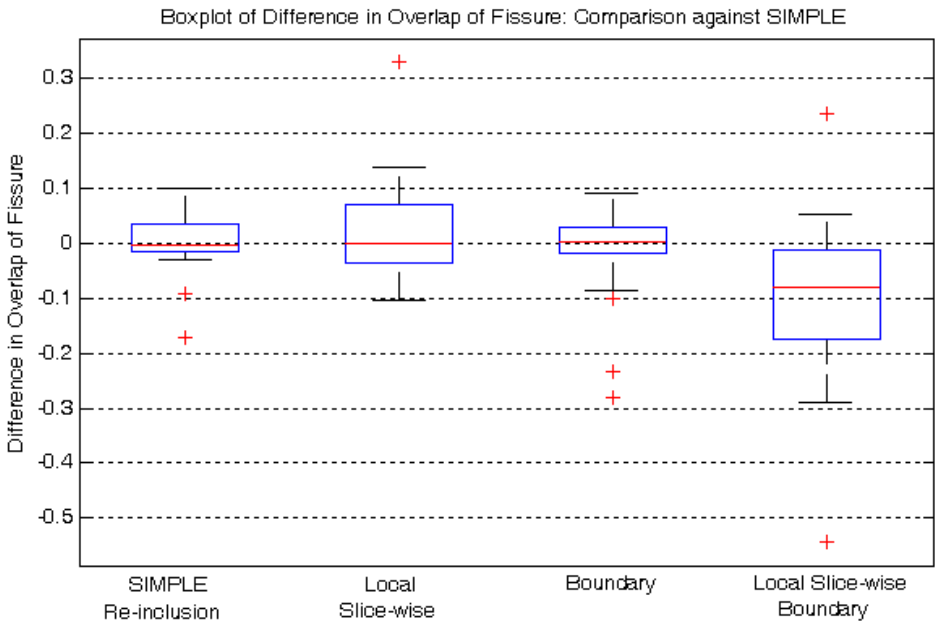


Figure 4.11: A box-plot of difference in overlap values of fissure.

between the fissure overlap values obtained from SIMPLE and 'SIMPLE with re-inclusion'. The test statistic comes out to be 0.7380 (see table 4.2) which means there is no statistically significant difference between the fissure overlap values from the two algorithms. On analyzing output from the re-inclusion algorithm, we realize that in most of the cases if re-inclusion algorithm rejects any input once, then this input is rejected in all the further iterations also. Therefore for majority of the cases, choice of re-including the inputs does not give any advantage over SIMPLE algorithm.

Local slice-wise SIMPLE algorithm: In figure 4.10, box-plot for 'local slice-wise' algorithm shows a bit higher median and entire box-plot is shifted slightly upwards compared to SIMPLE. Major part of box-plot for 'local slice-wise' algorithm is above 0 in figure 4.11, therefore it seems to be performing better compared to SIMPLE for majority of the examples. Interestingly, fissure overlap of one example is improve by more than 0.3 by slicewise algorithm, as shown in figure 4.11. This result encourages us to focus on *local* SIMPLE algorithms. The Wilkoxon test statistic between the fissure overlap values obtained from SIMPLE and slice-wise algorithm comes out to be 0.2355 (see table 4.2) which means there is no statistically significant difference between the fissure (3 voxel wide) overlap values from the two algorithms.

Table 4.2: Wilkoxon Test

algorithms	p-value
Global Re-inclusion	0.7380
Local Slice-Wise	0.2355
Global Boundary	0.7578
Local Slice-Wise Boundary	0.0106

Boundary SIMPLE algorithm: In figure 4.10, box-plot for 'boundary SIMPLE' algorithm looks just as good as that of the SIMPLE, although median is slightly higher but entire box-plot is shifted slightly downwards. A look at figure 4.11 tell us that boundary SIMPLE performs

worse for more examples than those for which it improves over SIMPLE. Two outliers below 0-line also testify worse performance of the boundary SIMPLE. The Wilkoxon test statistic between the fissure overlap values obtained from SIMPLE and 'local slice-wise'comes out to be 0.7578 (see table 4.2) which means there is no statistically significant difference between the fissure overlap values from the two algorithms. The suspect that the reason for such result is 2-fold:

1. If we look at figures 4.4 and 4.5, as the overlap of entire $3D$ input with corresponding ground truth increases, then overlap of boundaries of both the images also increases. Therefore, if SIMPLE rejects an input in first iteration because of low overlap of $3D$ volume, then boundary-SIMPLE also rejects that input because of low overlap value of $3D$ boundary. Therefore, boundary-SIMPLE does not improve over SIMPLE.

2. We expect boundary-SIMPLE to improve over SIMPLE since it enhances the sensitivity of SIMPLE. It gives a better estimate of how good an input segmentation is. But we have to keep in mind that we take entire boundary of an input segmentation into account for boundary-SIMPLE algorithm while the evaluation is performed only for fissure overlap. Therefore it does not add much to the sensitivity of SIMPLE.

Therefore, we think that boundary algorithm may help with an application where performance measure depends on entire boundary and not just fissure. Increasing sensitivity for such an application may be quite helpful.
Figure 4.12 shows the effect of changing boundary width on the performance of boundary SIMPLE algorithm. The change in boundary width has no significant effect on the performance.

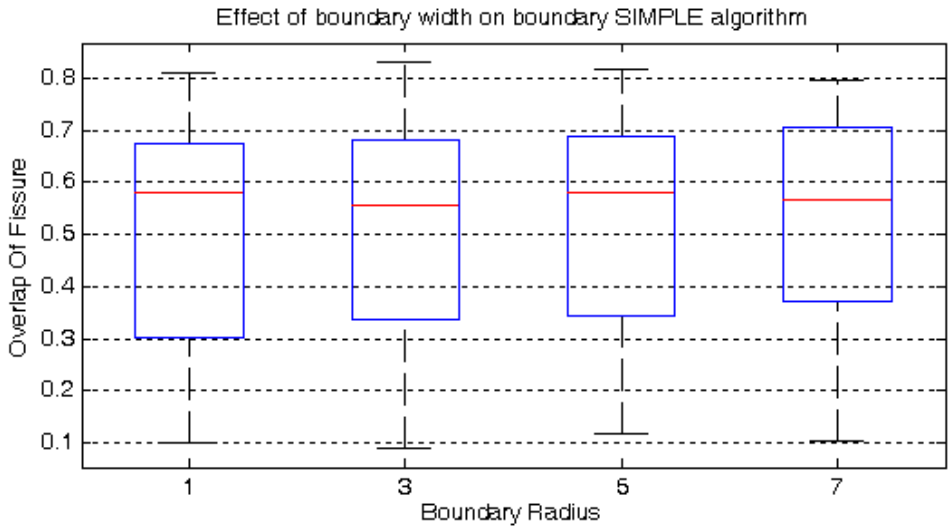


Figure 4.12: Boundary SIMPLE: effect of changing the boundary width.

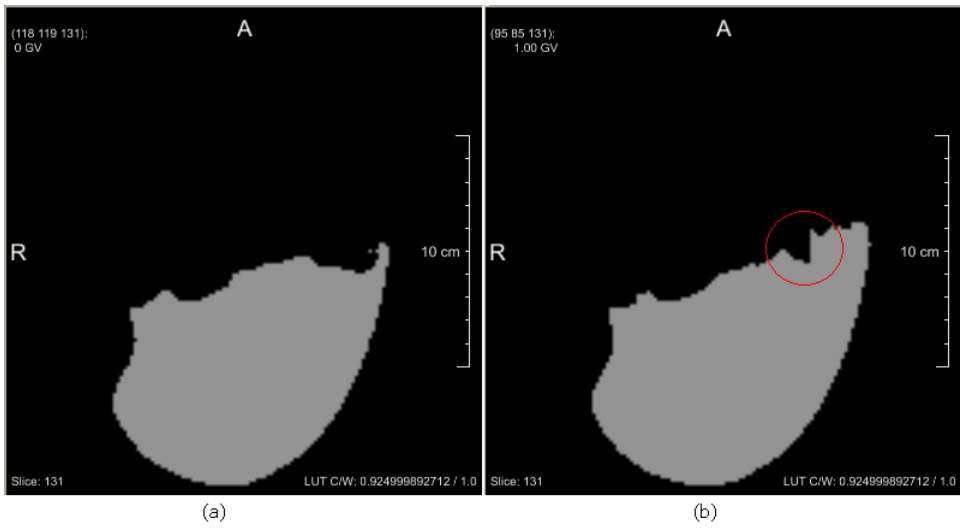
Local slice-wise Boundary SIMPLE algorithm: Local slice-wise Boundary SIMPLE algorithm performs worse compared to SIMPLE. Lowered box-plot in figure 4.10 and major part of the box-plot below 0-line in figure 4.11 confirm the same. The Wilkoxon test statistic confirms that SIMPLE is significantly better compared to local slice-wise Boundary SIMPLE algorithm. One possible reason for this may be because the boundary slice-wise algorithm computes weights by considering boundary of an input segmentation within a slice. In our opinion, this boundary contains too less voxels to measure DICE overlap reliably.

Local non-overlapping cube-wise SIMPLE algorithm (LNOC): Figure 4.13 shows results obtained on using cube-size 100 and cube-size 25 for local non-overlapping cube-wise algorithm. We can see a sudden jump in the fissure obtained on using cube-size 25. This *boundary-effect* may appear at *intersection-surfaces* of two cubes. Since final fissure within these two cube locations comes from separate weighted voting, performed on different set of cubes. Therefore, such jumps are possible. Following are the effects of changing the cube-size:

1. Smaller cube-size provides more local weights. The weights tend to be global on increasing the cube-size. Therefore, we lose the advantage of local fusion strategy.

2. As shown in figure 4.13, we may encounter jumps in fissure at intersection-surfaces of two cubes. A small cube-size may cause many intersection-surfaces within a fissure and consequently may cause many jumps. Therefore, a large cube size may provide more smooth fissure.

3. A very small cube-size may not contain enough voxels to compute DICE measure reliably.



(a) (b)

Figure 4.13: Results of local non-overlapping cube-wise SIMPLE algorithm on using cube-size (a)100 and (b)25. A sudden jump in the fissure can be observed in (b)

It is difficult to say how these 3 factors will interact with each other. Therefore we need to look at the performance of the LNOC algorithm to see which cube-size works better. In figure 4.14, the box-plot of fissure overlap is quite low for cube-size 10. It seems that cub-size 10 is too low (therefore contains too less voxels) to compute DICE measure reliably. For cube-sizes greater than 50, the algorithm performs better. Figure 4.15 confirms that cube-size 10 performance worse than SIMPLE, for most of the examples. One interesting observation in figure 4.15 is a positive outlier for cube-size 25 and higher. This particular example is greatly improved by LNOC algorithm. Figure 4.16 shows various slices for this outlier obtained from both SIMPLE and LNOC. One can see that fissure detected by both the algorithms is quite different in different local regions which accounts for such significant improvement for this example.
The performance of the algorithms vary arbitrarily at higher cube-sizes as shown in 4.15. We discussed the three effect of changing the cube-size on the performance. It is difficult to predict that which effect may dominate for a specified cube-size. The result of Wilkoxon test for

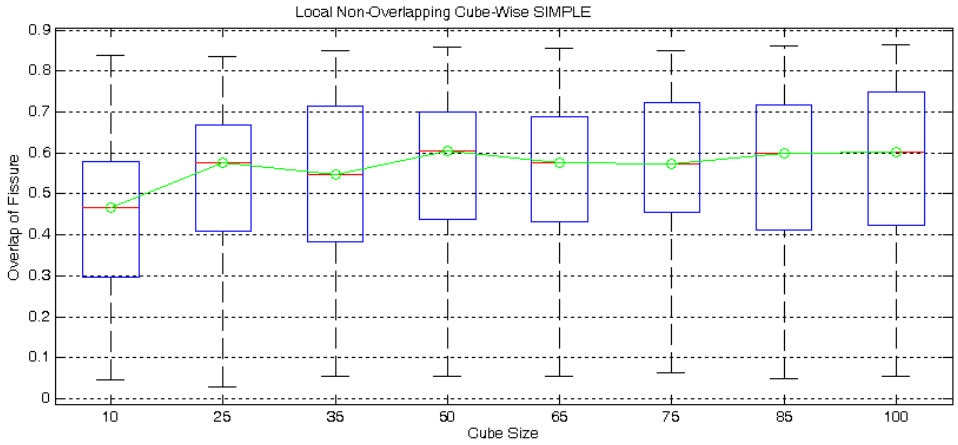


Figure 4.14: LNOC algorithm: Effect of changing cube size on fissure overlap for local non-overlapping cube-wise SIMPLE algorithm.

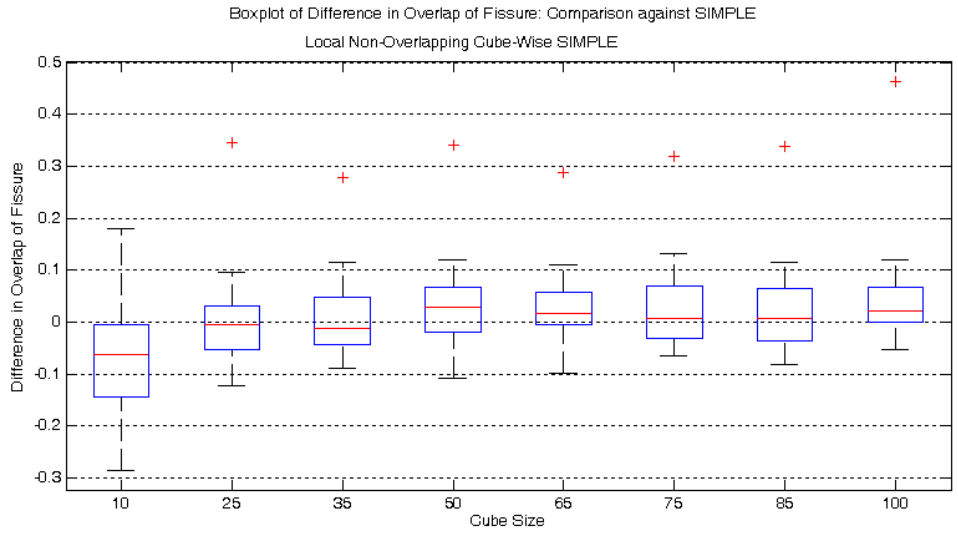


Figure 4.15: LNOC algorithm: Comparison against SIMPLE. Effect of changing cube size on difference in overlap of fissure for local non-overlapping cube-wise SIMPLE algorithm.

different cube-sizes is listed in table 4.3. The LNOC algorithm performs significantly better than SIMPLE for cube-size 100.

Local non-overlapping cube-wise SIMPLE with re-inclusion (LNOCR): LNOCR algorithm improves over SIMPLE for some examples (see figure 4.18), but also deteriorate fissure overlap for almost equal number of examples. The Wilkoxon test statistic confirms (table 4.3) that the LNOCR does not provide any statistically significant improvement over SIMPLE. The reason for such performance is same as discussed for 'global SIMPLE with re-inclusion' algorithm.

Local non-overlapping cube-wise boundary SIMPLE algorithm (LNOCB): Figure 4.19 and 4.20 give an overview of the performance of LNOCB algorithm for different cube-sizes. The algorithm shows no significant improvement over SIMPLE (table 4.3). Since local boundary algorithms hold too less information in a local region to compute the DICE measure reliably. The reasons responsible for poor performance of boundary SIMPLE algorithm play role here as well.

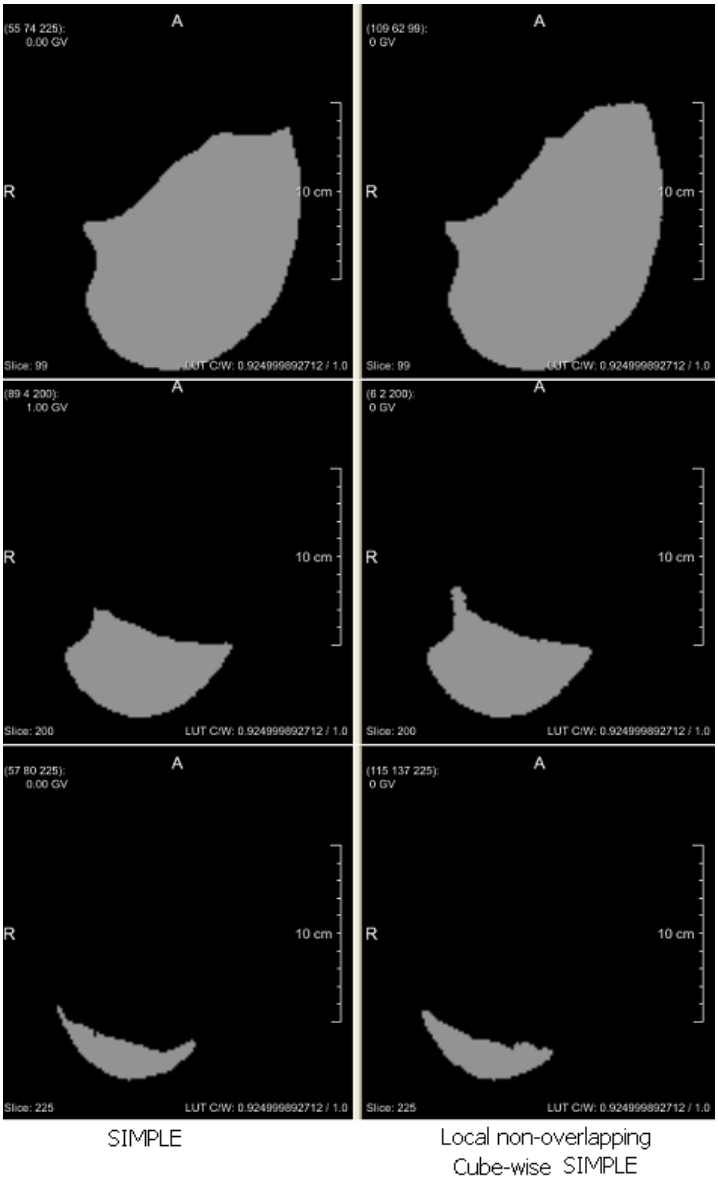


Figure 4.16: Various slices of a resultant fusion image obtained from SIMPLE and local non-overlapping cube-wise SIMPLE algorithm. The fissure overlap for this example is improved by approximately 0.3 compared to SIMPLE.

Local overlapping cube-wise SIMPLE algorithm (LOC): Figure 4.21 and 4.22 give an overview of performance of LOC algorithm for different cube-sizes. The algorithm shows statistically significant improvement over SIMPLE for cube-size 50 and higher (table 4.3).
One point must be noted that LOC algorithm is devoid of boundary-effect, which contributes to improved performance. Also the smoothing effect, discussed in section 4.2.1.3, helps in improving the performance. We can see in figure 4.22 that LOC consistently performs better than SIMPLE, therefore it gives us freedom to select any cube-size greater than 35 for an unseen image. Since a higher cube-size increases computation time of LOC algorithm, therefore one should prefer a cube size close to 65 for better and fast results. Remember that this optimum cube-size may be different for some other data set containing inputs of different dimensions than ours.

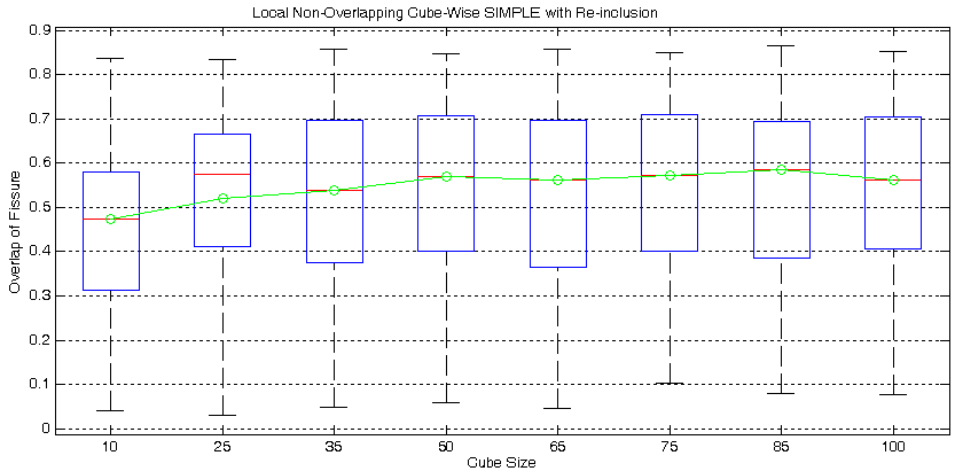


Figure 4.17: LNOCR algorithm: Effect of changing cube size on fissure overlap for local non-overlapping cube-wise SIMPLE with re-inclusion algorithm.

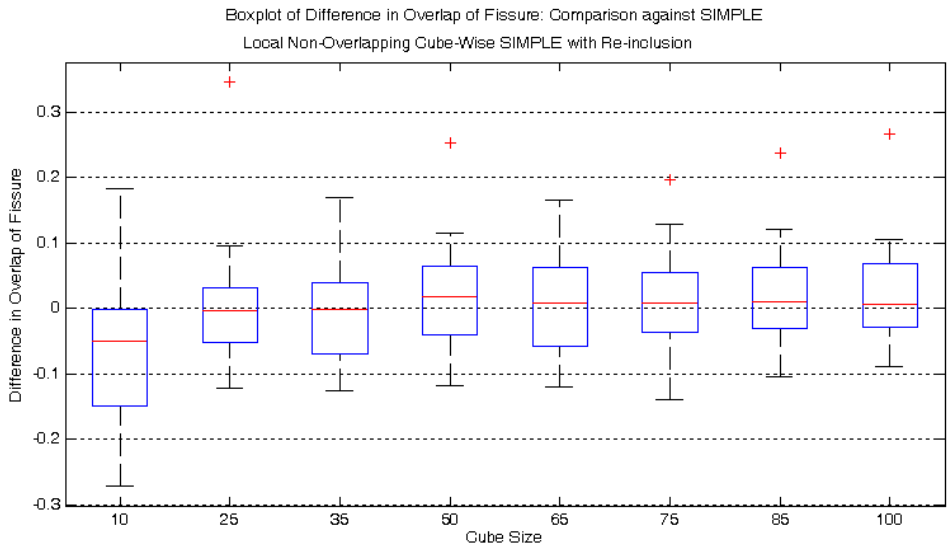


Figure 4.18: LNOCR algorithm: Comparison against SIMPLE. Effect of changing cube size on difference in overlap of fissure for local non-overlapping cube-wise SIMPLE with re-inclusion algorithm.

Table 4.3: Wilkoxon test statistics for local cube-wise SIMPLE algorithms, Threshold = 0.05

Cube Size	LNOC	LNOCR	LNOCB	LOC
10	0.0039	0.0042	0.0026	0.0014
25	0.8552	0.8552	0.2604	0.5841
35	0.8314	0.5230	0.5841	0.1443
50	0.0516	0.3778	0.3155	0.0386
65	0.1283	0.6265	0.7380	0.0177
75	0.3458	0.4115	0.5304	0.0089
85	0.3458	0.4842	0.6702	0.0106
100	0.0042	0.3304	0.1209	0.0056

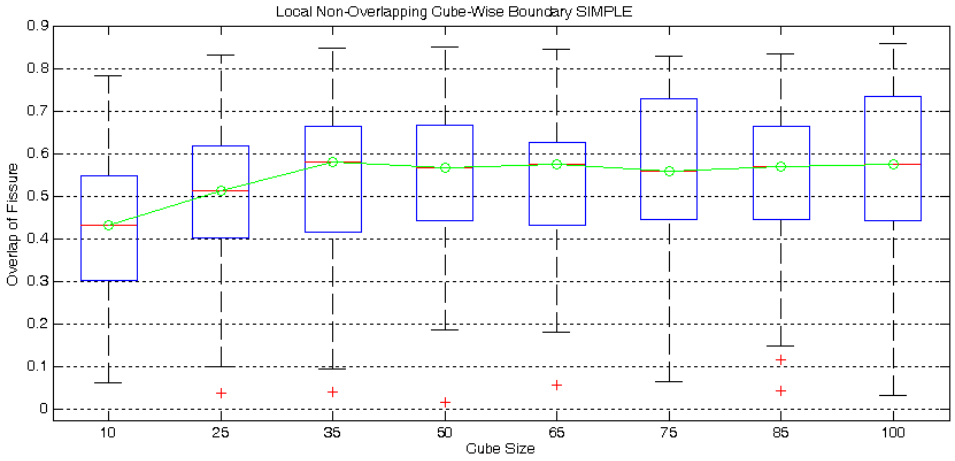


Figure 4.19: LNOC algorithm: Effect of changing cube size on fissure overlap for local non-overlapping cube-wise boundary SIMPLE algorithm.

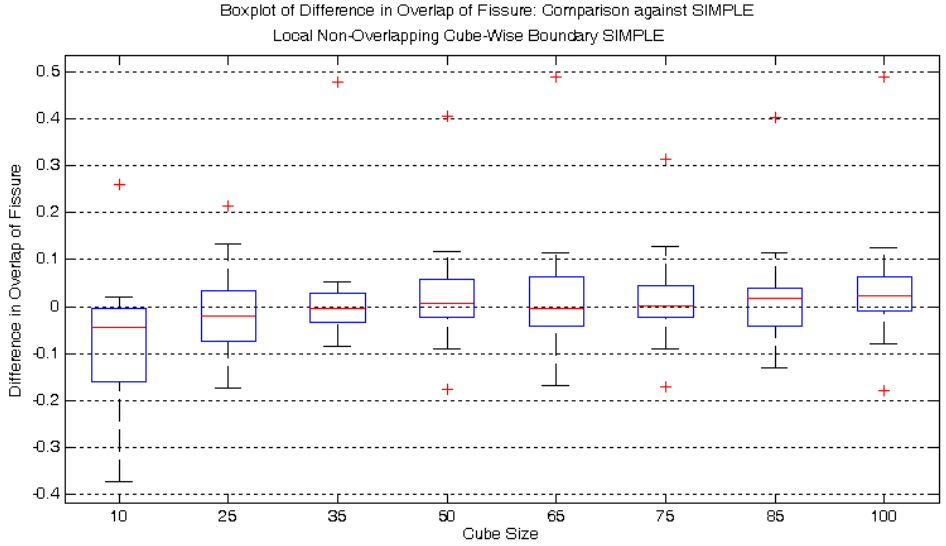


Figure 4.20: LNOC algorithm: Comparison against SIMPLE. Effect of changing cube size on difference in overlap of fissure for local non-overlapping cube-wise boundary SIMPLE algorithm.

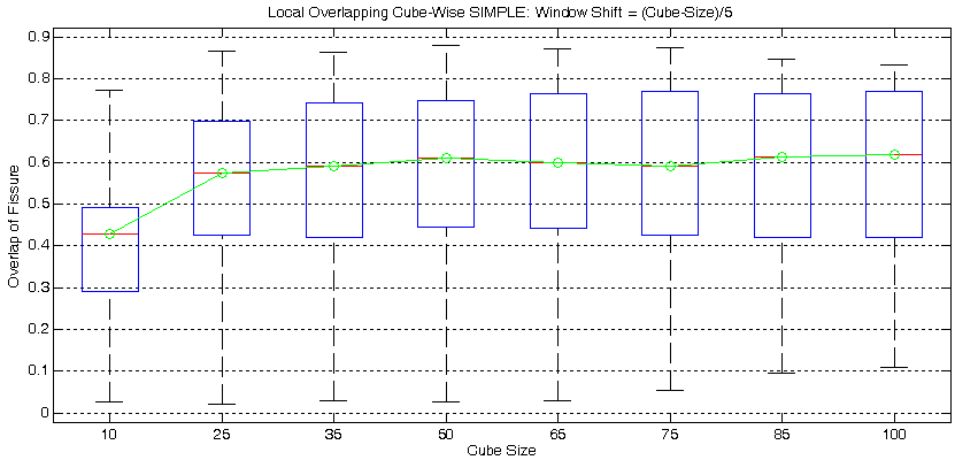


Figure 4.21: LOC algorithm:Effect of changing cube size on fissure overlap for local overlapping cube-wise SIMPLE algorithm.

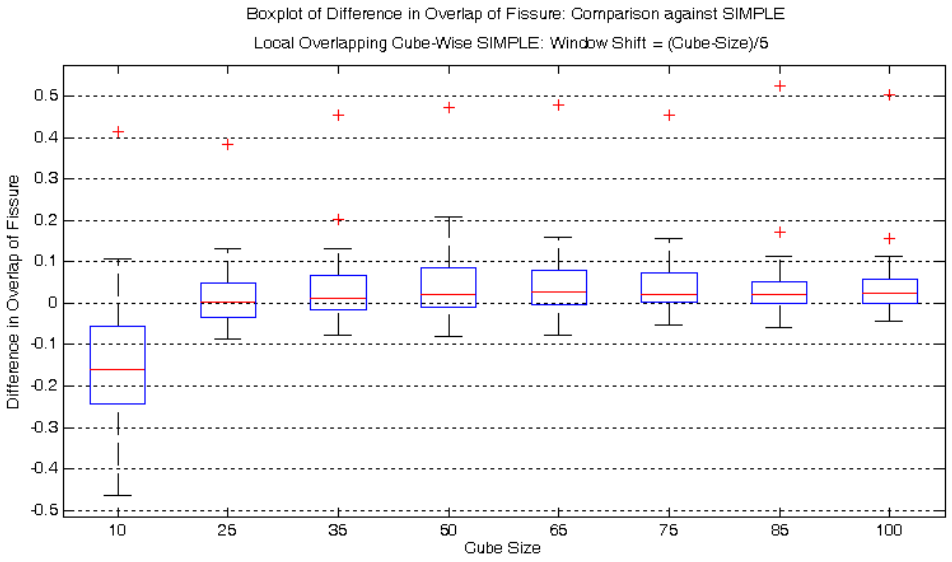


Figure 4.22: LOC algorithm:Comparison against SIMPLE. Effect of changing cube size on difference in overlap of fissure for local overlapping cube-wise SIMPLE algorithm.

Multi-Label SIMPLE algorithm: Figure 4.23 shows the performance of non-overlapping cube-wise multi-label SIMPLE and figure 4.23 for overlapping cube-wise multi-label algorithm. As we saw in the case of binary counterparts of these algorithms (figure 4.14 and 4.21), the performance of the overlapping algorithm is consistent on changing the cube size. A set of optimum α values, for multi-label algorithms, is given in the appendix 9.4.

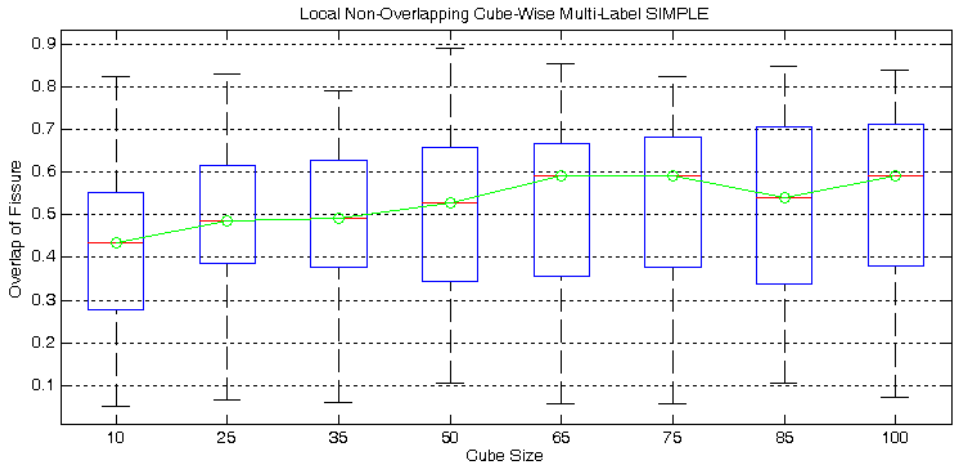


Figure 4.23: Effect of changing cube size on the fissure overlap for local non-overlapping cube-wise multi-label SIMPLE algorithm.

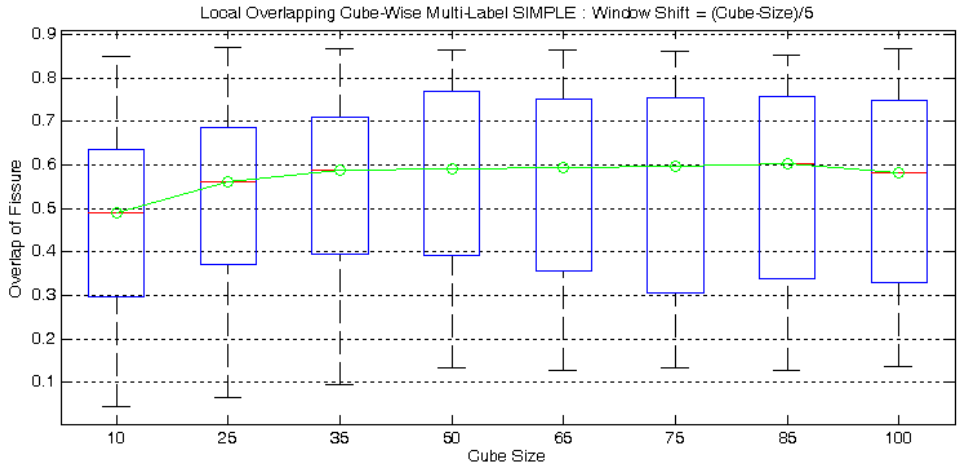


Figure 4.24: Effect of changing cube size on the fissure overlap for local overlapping cube-wise multi-label SIMPLE algorithm.

Comparison between the results from binary and multilabel SIMPLE The multi-label version of the overlapping cube-wise algorithm also provides significant improvement over the results of binary global SIMPLE, as shown in table 4.4. Our notion that cube wise 10 is too small to hold meaningful information for DICE calculation is again confirmed by the Wilkoxon test statistics in the table 4.4. The performance of multi-label overlapping algorithm is consistently better that SIMPLE for all the cube sizes greater than 25. Figure 4.25 shows box-plot of difference in fissure overlap values between multi-label overlapping and binary global SIMPLE.

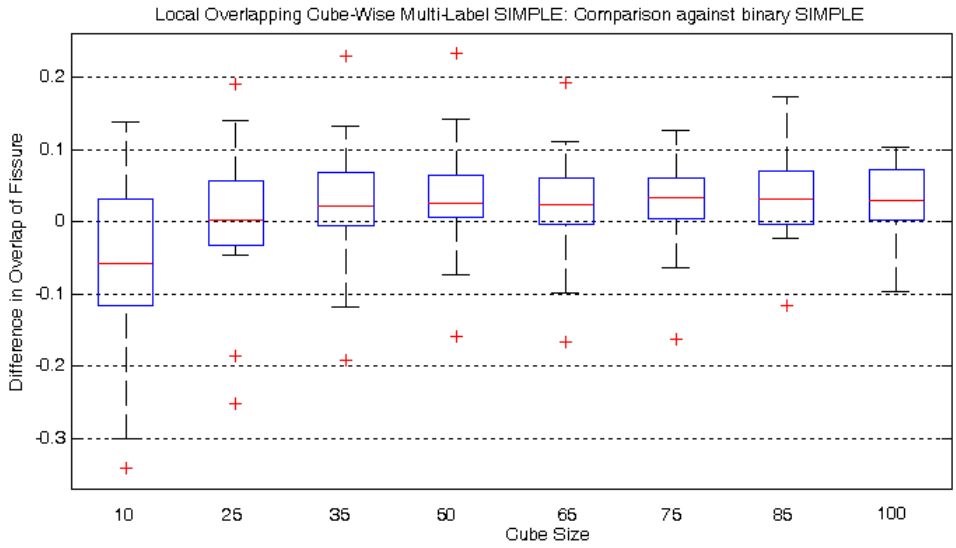


Figure 4.25: Comparison against SIMPLE. Effect of changing cube size on difference in overlap of fissure for local overlapping cube-wise multi-label SIMPLE versus binary global SIMPLE algorithm.

Table 4.4: Wilkoxon test statistics for local overlapping cube-wise multi-label versus global binary-SIMPLE, Threshold = 0.05

Cube Size	LOC
10	0.0516
25	0.4115
35	0.0480
50	0.0177
65	0.0138
75	0.0208
85	0.0081
100	0.0068

More results on the comparison between multi-label algorithms and their binary counterparts, are given in the appendix 9.5.1. The performance of a multi-label algorithm is not exactly the same as its binary version. We are not able to explain the reason of such result. It would be interesting to study it in detail. There was no statistically significant difference found in the performance of a multi-label algorithm and its binary version.

4.4 Conclusion

The aim of multi-atlas based segmentation is to cope for the inaccuracies of individual ABSs. We obtain several estimate of segmentation of an unseen image through independent ABSs. The task comes down to combining these input segmentations to form one final estimate of segmentation of the unseen image. SIMPLE performs this by weighted voting of inputs, where weights are a global DICE overlap value. For our data, it is most useful to focus on a combination strategy which is able to select the locally good regions from different segmentations. The methodology of computing *local* goodness of an input segmentation is crucial for the success of a local combination scheme. We used SIMPLE algorithm as basis and proposed several methods for estimation of local goodness of inputs. In this chapter we focus on making a separate local decision for different regions within inputs. We do not change the weights used, i.e. DICE overlap, and the fusion criteria, i.e. weighted voting, as is used in the SIMPLE algorithm.

An input segmentation can be divided into different local regions in many different ways. We experimented with 3 of them: slice-wise, non-overlapping cube-wise, and overlapping cube-wise. Slice-wise algorithm does not show any statistically significant improvement over SIMPLE. We experimented considering slices to be stacked over each other along Z-dimension only because of time constraints, it would be interesting to see if the performance changes on considering slices to be arranged along any other dimension. In our next local fusion approach, non-overlapping cube-wise algorithm seems to work significantly better than SIMPLE for some particular cube-sizes such as 50 and 100. For these particular cube-sizes, the local cubes divide the fissure within input segmentations is intersected in such a way that it produces least boundary effect (see section 4.3 for details on boundary effect). Now, the problem is that these two cube sizes work best for our data set but we do not know which cube size may perform well for an unseen CT scan image. We propose overlapping cube-wise algorithm to address this problem. As the name suggests, we divide an input image into overlapping cubes in this algorithm, which provides a smooth change in local weights because of averaging effect. Also, sharing of a large number of voxel between adjacent cubes ensure that boundary-effect is minimized. As a result of this, overlapping cube-wise algorithm shows significant improvement over SIMPLE for any cube-size greater than 35. Therefore one should prefer using cube-size 50 or 65, since the computation time increases for higher cube-sizes. It is found in our experiment that DICE overlap does not give reliable weights for a cube size less than 25 or lesser.

We experimented with several global and local boundary algorithms with focus on increasing the sensitivity of SIMPLE for rejection (or selection). Boundary algorithm considers overlap of only boundary of an input segmentation with that of the estimation of ground truth. As is shown in figure 4.5, this algorithm takes overlap of entire boundary of an input segmentation into account, but we evaluate the segmentation performance only by measuring the fissure overlap. The remaining boundary matches almost perfectly for all the inputs. Therefore, it does not add much sensitivity to SIMPLE. We recommend to use boundary SIMPLE algorithm only for applications where we expect segmentation error at the entire boundary of an object. On the other hand, all the local boundary algorithms: slice-wise and cube-wise perform worse compared to SIMPLE. We believe that boundaries within these local regions contain too less voxels to compute DICE overlap reliably. Therefore weights are not representative of goodness of an input segmentation. We infer that local boundary algorithms are not a good choice.

Once rejected, SIMPLE never uses any input back in the final estimation of ground truth. We suspect that some of the inputs may be rejected too early and their weights may improve over iterations as the estimation of ground truth improves. Our experiments confirms this notion.

In some cases inputs are indeed re-included in the fusion process after getting rejected at initial iterations. But for majority of the cases if an input is rejected in an initial iteration, its rejected in all the future iterations. Therefore, we do not see any significant improvement over SIMPLE on enabling it to re-include the rejected data.
SIMPLE algorithm works only for binary segmentation. Since our data set contains 3 labels for left lung and 4 labels for right lung, we felt the need of a multi-label SIMPLE algorithm. Although experimented with few other multi-label segmentation fusion algorithms, but multi-label SIMPLE is a significant improvement over them. Figure 4.8 shows the performance of majority voting and STAPLE on our dataset. SIMPLE and thus, multi-label SIMPLE gives significant improvement over others. We proposed a global and few local multi-label algorithms. As we already saw in the case of binary inputs, overlapping cube-wise algorithm works best for multi-label data as well and it performs well for cube sizes greater than 25. Appendix-9.5.1 shows some more results on a comparison among several multi-label versus binary algorithms.
This section would be incomplete without discussing on α used in the threshold calculation. α controls the amount of data that can be thrown in overall process. In a way it also controls the convergence speed of an algorithm. A particular α may allow more than half of the data to be thrown while some other α may not allow any rejection at all. SIMPLE was tested on a dataset containing 100 input segmentations. Therefore even if some α ends up throwing 75% of the data, they are still left with 25 good segmentations. For the same reason choice of α does not have any significant effect on the performance for their dataset. In our case, choice of α is important since we have only 22 inputs. We experimented a number of α values for each algorithm and a set of six α is recommended for each algorithm. There was no significant difference in performance of an algorithm for any of these six α. Since we have relatively small dataset, the best α are those which do not reject much data. In our case as α which stops at 15-18 inputs (for global algorithms), starting from 23 works best. One reason responsible for the performance of re-inclusion algorithm is the fact that we already tune the algorithms to α values which reject very less data, therefore using the feature of re-inclusion does not improve. One look at the α values given in the appendix 9.4 helps in understanding the α which works well for all the algorithms. $\alpha = 1/f$ works well for all of our good performing algorithms such as binary and multi-label local overlapping cube-wise and global multi-label. The same set of α values was used for both: the left and the right lung. Also, these α values are optimum for both the registration setups: A and B. Therefore, we conclude that choice of α is specific to an algorithm and $\alpha = 1/f$ is a good choice for all the good performing algorithms. f-value denotes fraction of selected segmentations in previous iteration.

Chapter 5

Improving the Input Segmentations via Image Registration

This chapter contains discussion and results of various segmentation fusion strategies discussed in chapter-4, on using registration setup-B for each atlas based segmentation. The results are presented for both the left and the right lung. For left lung, overlap of 3 voxel wide horizontal fissure is computed, from the segmentation fusion output. In the case of right lung, results for both the oblique and the horizontal fissures are provided. Since the right lung contains 4 labels as compared to the 3 labels present in the left lung (see figure 5.1), we used only multi-label SIMPLE algorithms for locating fissures in the right lung. In case of left lung, we used binary input as shown in the figure 4.2.

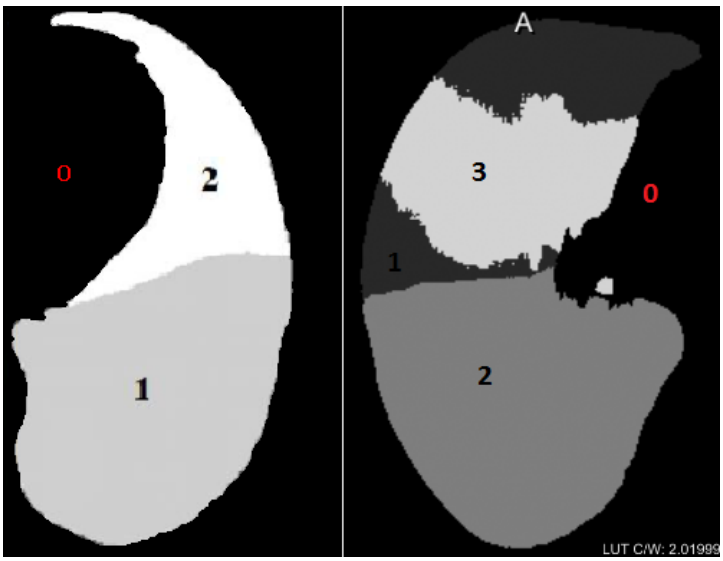


Figure 5.1: (Left) 3 labels present in the manual segmentation of the left lung. (Right) 4 labels for the right lung.

In the chapter 4, we discussed various improvements over SIMPLE. The best result that we achieve is by 'local overlapping cube-wise SIMPLE' algorithm (figure 4.21), but it is noticeable that even the best result obtained is far less accurate than manual segmentation. The reason becomes clear if we give a look to some of the input segmentations (i.e. DS_i) for a particular fixed image. See figure 5.2. All the input segmentations shown miss the correct fissure almost completely. It is not possible for any segmentation fusion method to make a good estimate of

fissure on combining these inputs. Therefore, we need to improve the fissures present in the input segmentations in order to obtain more accurate fissure. Since each input segmentation is obtained through an ABS process, as discussed in section 2.2. Therefore, we need to improve the registration for obtaining better inputs, i.e., those with fissure more similar to that in the ground truth. This time we use registration setup-B, discussed in section 3.3.4.2.

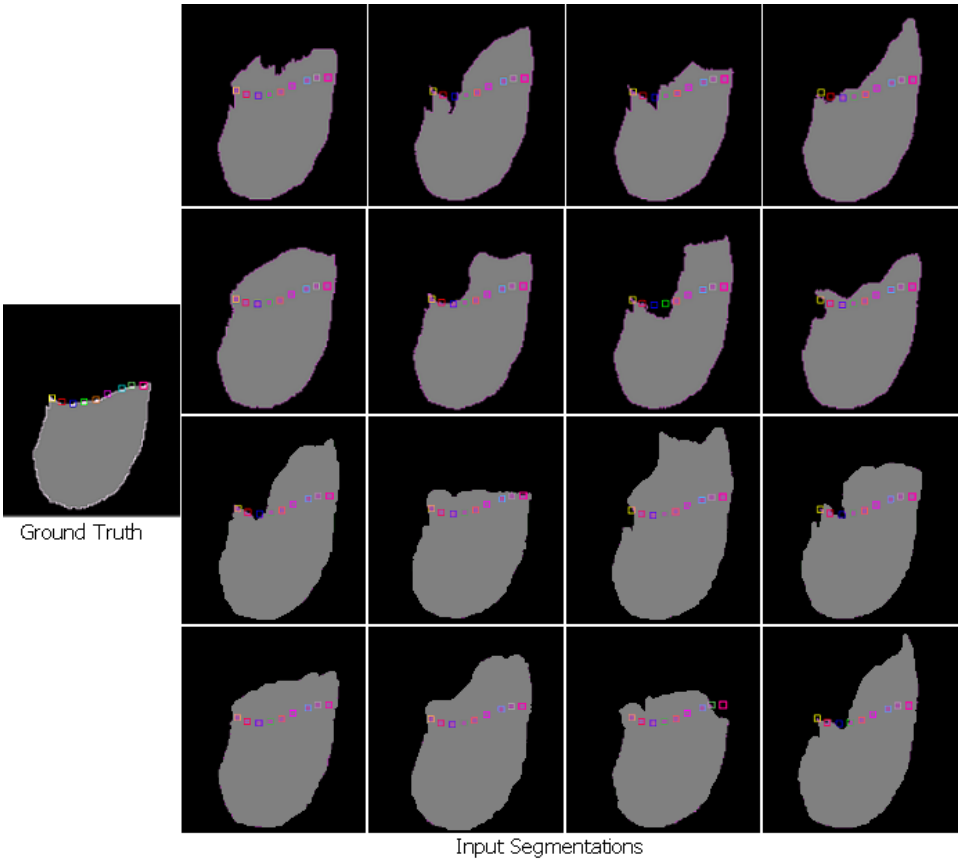


Figure 5.2: Registration setup-A: Some of the bad input segmentations are shown for a particular fixed image. Some points on the fissure in ground truth image are selected and same points are shown in input segmentations. It is clear that all the input segmentations shown miss the fissure almost completely. No fusion criteria can give a good estimate of ground truth fissure for such input set.

5.1 Results and Discussion for Left Lung

Figure 5.3 shows the overlap of 3 voxel wide fissure obtained from various global and local binary SIMPLE algorithms discussed in chapter 4. The overlap of fissure in unexpectedly high and all the algorithms seem to be performing equally well. If we look at the boxplot of difference in overlap of fissure given in figure 5.4 and 5.5, we realize that there is no significant difference in the overlap of fissure obtained from any algorithm. These results are almost as good as manual segmentation. Let's give a look to the input segmentations. Figure 5.6 shows some of the input segmentations obtained on using registration setup-B for atlas based segmentations. The fixed image is same for both the figure 5.2 and 5.6. As it turns out, the registration setup-B creates good fissures in all the input segmentation. A strategy as simple as majority voting is able to account for the slight disagreement among various inputs.

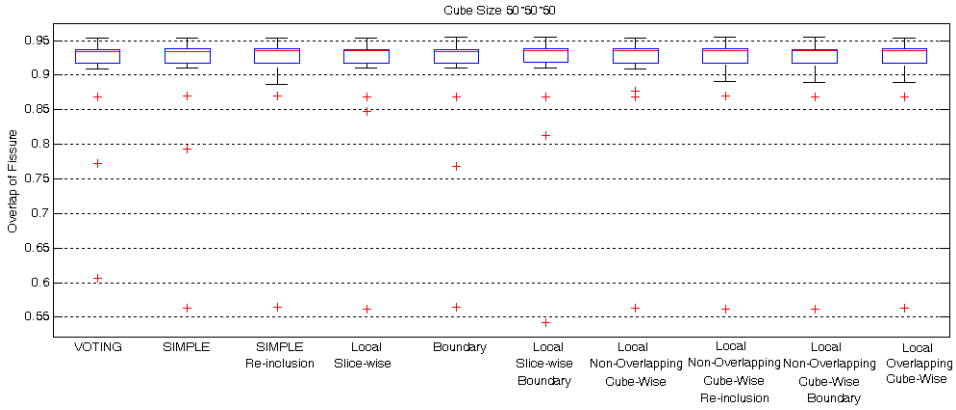


Figure 5.3: Left lung: Box-plot of the 23 overlap values obtained on comparing fissures obtained from ground truth and various SIMPLE algorithms.

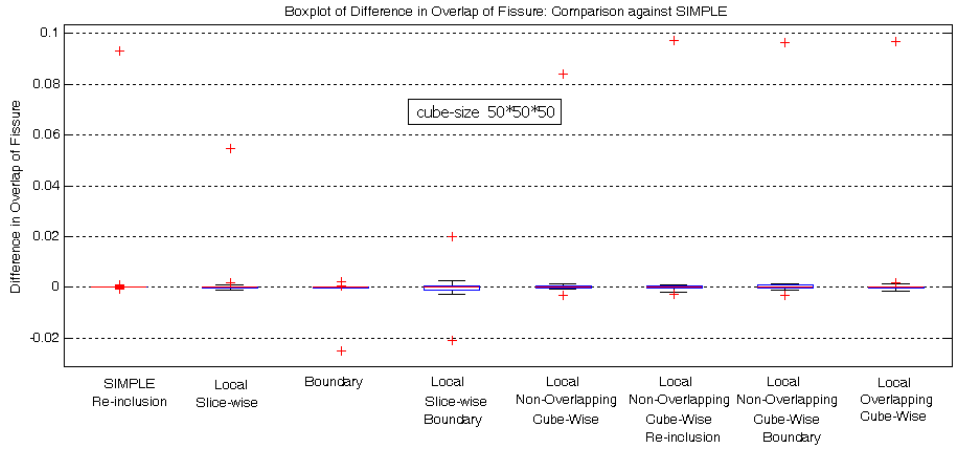


Figure 5.4: Left lung: A box-plot of difference in overlap values of fissure on comparing against SIMPLE.

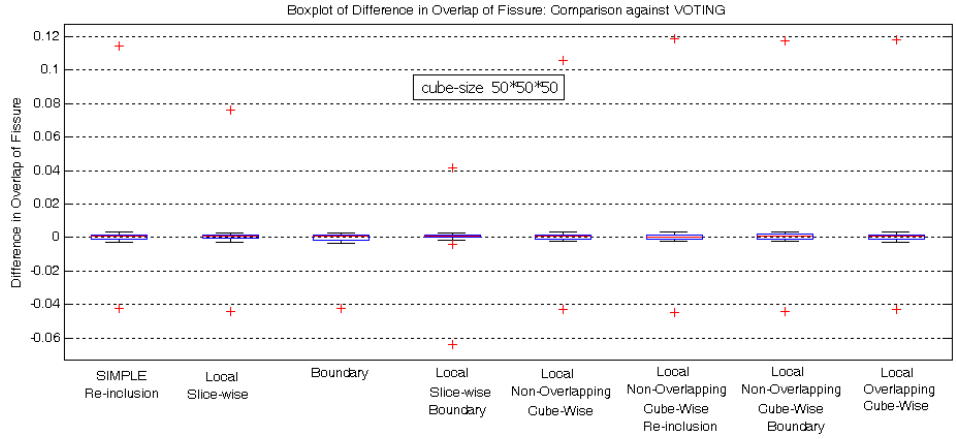


Figure 5.5: Left lung: A box-plot of difference in overlap values of fissure on comparing against majority voting.

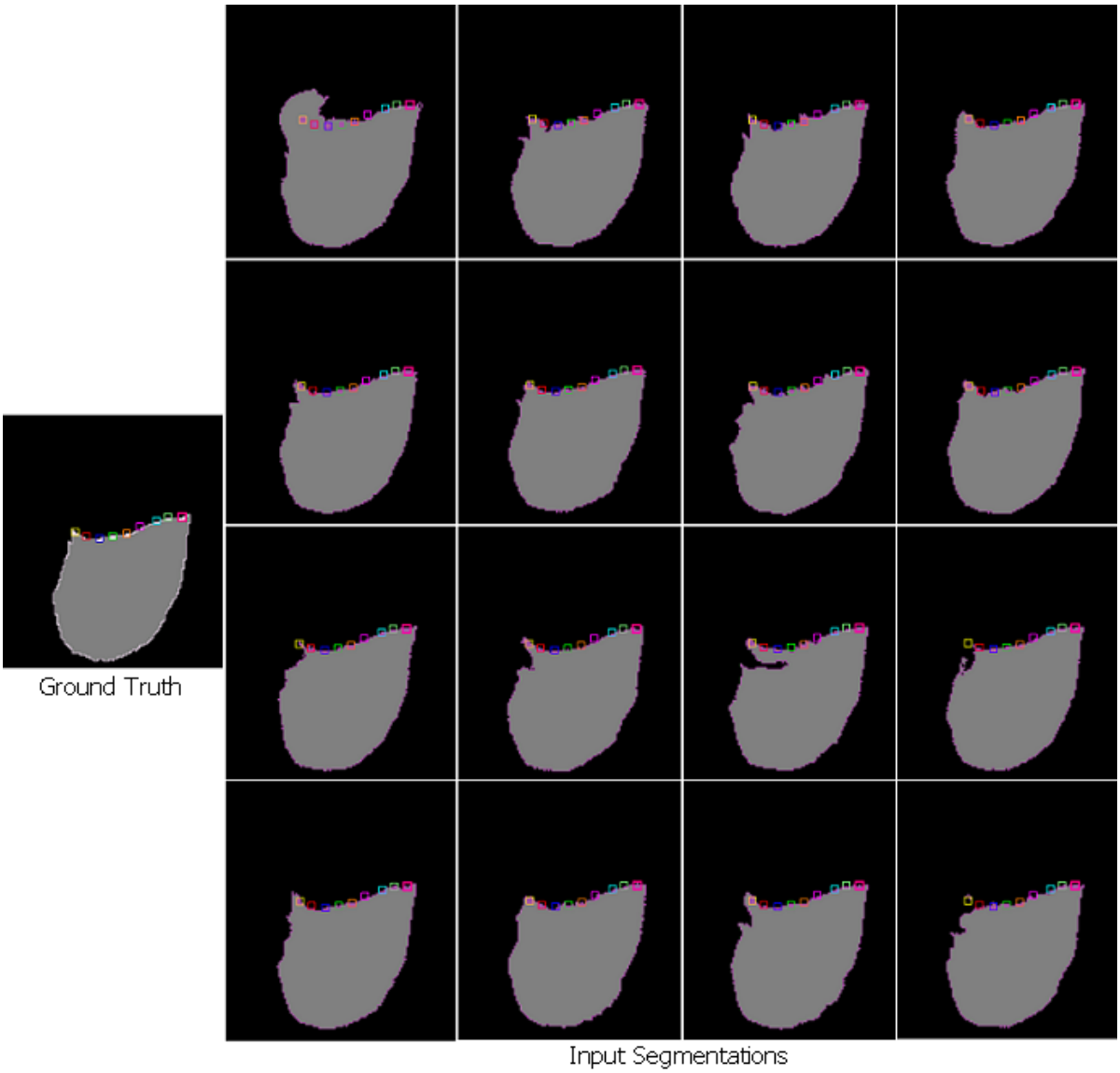


Figure 5.6: Registration setup-B: Some of the bad input segmentations are shown for a particular fixed image. Some points on the fissure in ground truth image are selected and same points are shown in input segmentations. It is clear that all the input segmentations shown miss the fissure almost completely. No fusion criteria can give a good estimate of ground truth fissure for such input set.

5.2 Results and Discussion for Right Lung

We saw that registration setup-B is able to generate almost perfect fissure for left lung. It is close to the manual segmentation that we used. We decided to use the setup-B for right lung too. Presence of two fissures makes it a more difficult registration problem as compared the the left lung, therefore we expect lower performance for right lung. Figure 5.7 and 5.8 show the result obtained for the horizontal and the oblique fissure respectively. The results are again very close to the manual segmentation. All the algorithms perform equally good here.

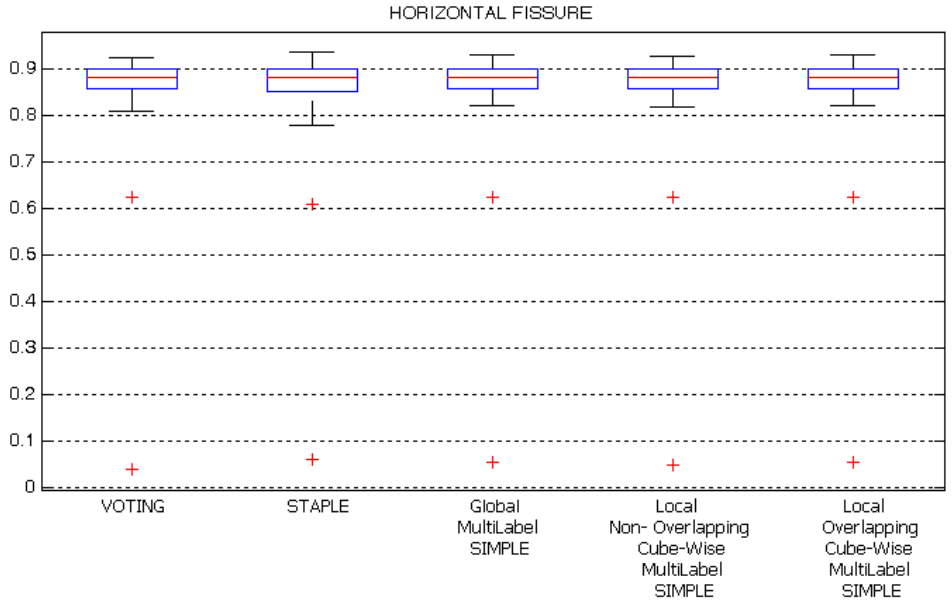


Figure 5.7: Right lung: Box-plot of the 23 overlap values obtained on comparing fissures obtained from ground truth and various SIMPLE algorithms.

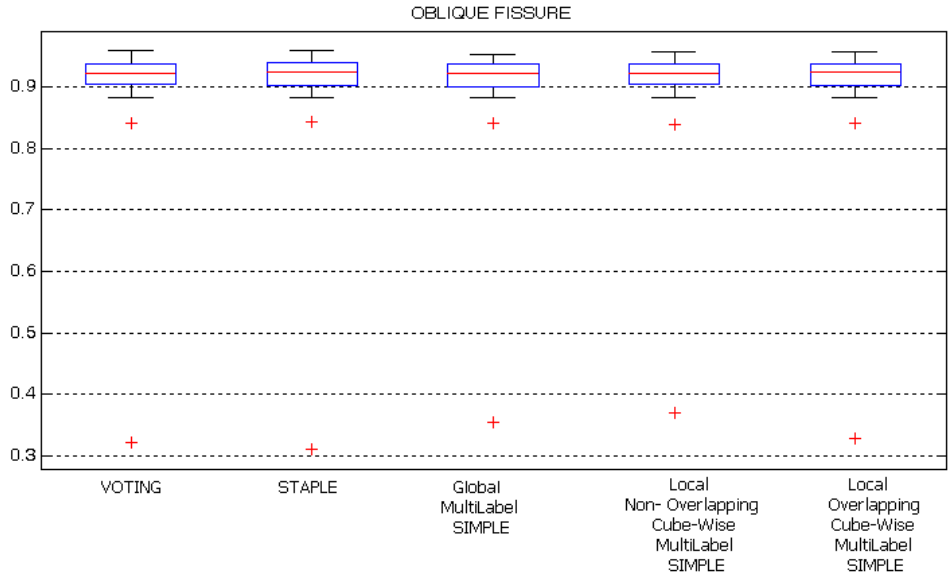


Figure 5.8: Right lung: Box-plot of the 23 overlap values obtained on comparing fissures obtained from ground truth and various SIMPLE algorithms.

5.3 Conclusion

If most of the input segmentations are wrong at a particular local region, then most probably their fusion output may also be wrong at the location. The only way to improve this situation is to obtain better input segmentations. We experimented with registration setup-B with the same motivation. Quite to our surprise the registration improves the input segmentation to great extent. For such a good input set we do not even need to do MABS, since a single ABS itself can achieve the accuracy of manual segmentation. We consider the right lung as a more difficult data to register compared to left lung, since alignment of both the horizontal and the oblique fissure is required from a moving to the fixed image. But we again see almost perfect registration. It would not be wrong to say that the problem of lobe segmentation is resolved to the accuracy of manual segmentation. We still find the registration setup-A useful, the one which created poor segmentations, since inaccuracy of input segmentations makes it a difficult segmentation fusion problem. We require locally good input set to test our various algorithms. Each algorithm performs almost equally good on using the inputs generated from setup-B, therefore it is not possible to do comparative study of algorithms for such an ideal input set. It is quite interesting to see how quality of registration can drastically affect the segmentation fusion output.

Chapter 6

Fixed Rules: Performance Based on Success of Registration

Consider an atlas based segmentation process in which a moving image M_i is registered to a fixed image F. See figure 6.1. In addition to the deformed moving image DM_i, a transform T_i from the moving image to the fixed image is also obtained. On applying this transform T_i on the segmentation of M_i, i.e. S_i, deformed segmentation DS_i is obtained. See figure 6.2. This DS_i is an estimate of segmentation of the fixed image F. If registration was perfect, the deformed moving image DM_i would be exactly same as the fixed image F. Also, deformed segmentation DS_i would be exactly same as segmentation of the fixed image, in case of a perfect registration. It can be concluded that a better registration results into higher similarity between the deformed moving image DM_i and the fixed image F. In such a case deformed segmentation DS_i would be more similar to the segmentation of fixed image.

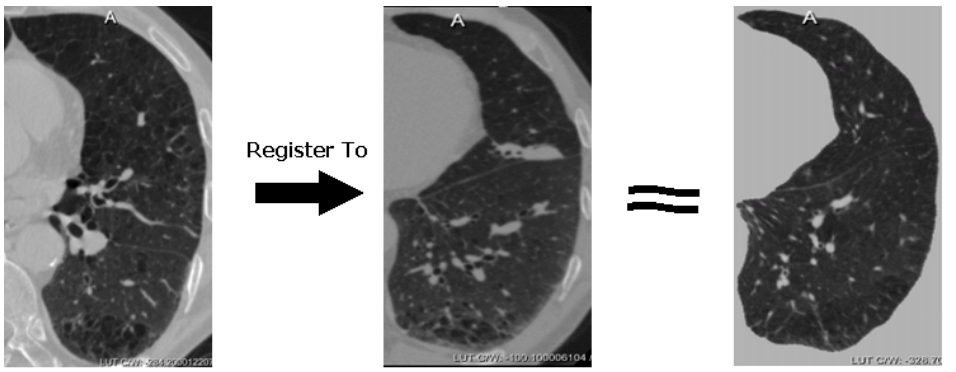


Figure 6.1: Moving image M_i is registered to fixed image F. As a result of this registration deformed moving image DM_i and a transform T from the moving image to the fixed image is obtained.

In the current chapter, our first focus is on finding local weights for a deformed segmentation DS_i based on the similarity between DM_i and F, which eventually depends on the 'local success of registration'. Higher similarity between deformed moving image DM_i and the fixed image F in a particular local regions results into higher weight for corresponding deformed segmentation DS_i, for that local region. Secondly, we aim to research on finding a suitable fusion strategy for the local weights found.
In the chapter 4, we found local weights based on the accuracy of the input segmentations DS_i.

Although accuracy of input segmentations depends on the success of registration. But in this chapter we make direct use of the registration quality for finding the local weights for the input segmentations. Also, in the chapter 4 we used weighted voting as the fusion criteria for all the algorithms. In this chapter we experiment with several *fixed rules* as fusion strategies. As such weighted voting itself is kind of a fixed rule since it behaves in a fixed manner irrespective of the input data. But we have also experimented with other fixed rules such as summation, median, maximum, product etc. Performance of fixed rules depends on the input dataset. It is hard to tell beforehand that which fixed rule may work better than others for our dataset. Therefore, we also aim to find the most suitable fixed rule for our application.

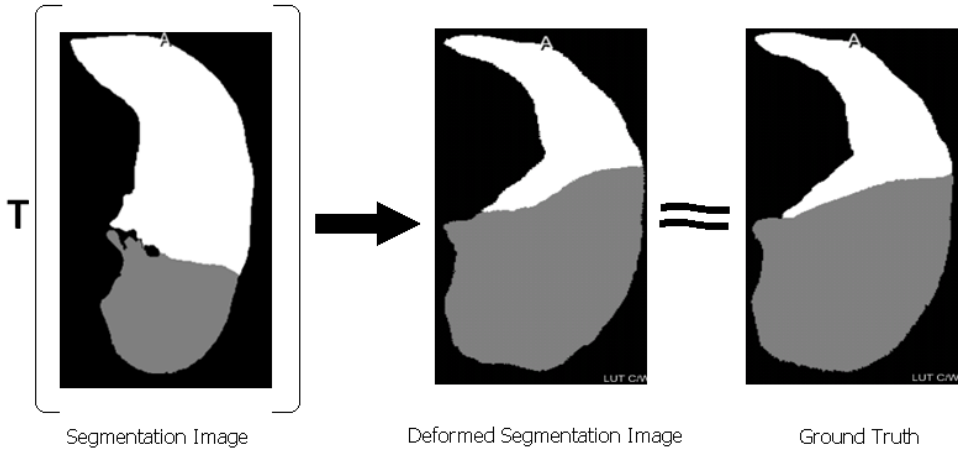


Figure 6.2: Transform T obtained from previous registration step is applied on the segmentation S_i of moving image M_i. Deformed segmentation DS_i is obtained. DS_i is an estimate of segmentation of fixed image F, i.e., ground truth. If the registration is perfect, DS_i would be exactly same as segmentation of fixed image.

This chapter contains results generated only for left lung. All the input segmentations were obtained through multi-atlas based segmentation and setup-A (see section 3.3.4.1 for details) was used for performing all the registrations. In the section sec:weightgeneration, the theory of finding local weights based on the success of registration is explained. We propose to compute *confidence values* from these weights, whose importance and methodology is explained in the section 6.2 and 6.3. Section 6.4 gives detailed of the parameters used in the proposed setup. These parameters need to be tuned for our dataset. The theory of fixed rules and their working is explained in the section 6.5. Since we found the computation time of our algorithm to be quite high, we use a *trick* to improve the run time of our algorithm. This trick is explained in the section 6.6. Section 6.7 contains results and discussion. Section 6.8 concludes.

6.1 Weight Generation

As was mentioned in section 2.4.4, if all the segmentations DS_i are not equally good, then one should assign higher weight to a more accurate segmentation. Artaechevarria et al. [2009] and I. Isgum and van Ginneken [2009] used success of registration as base for calculating weights for individual segmentations.

6.1.1 Using distance map of CT data for weights

We propose to use distance map of CT scan images for calculating weights. First of all we make an initial estimate of fissure for both fixed and moving images. For fixed image, we directly compute the distance transform on this initially estimated fissure, as shown in figure 6.3. Let's call this distance map $dist_F$. While for a moving image, we make use of the fissure present in the corresponding deformed moving image DM_i. The fissure is extracted from the deformed moving CT image by intensity thresholding, and then distance transform of that fissure is computed. Let's call it $dist_{M_i}$. See section 3.3.4.1 for details of the computation of the initial estimate of fissure.

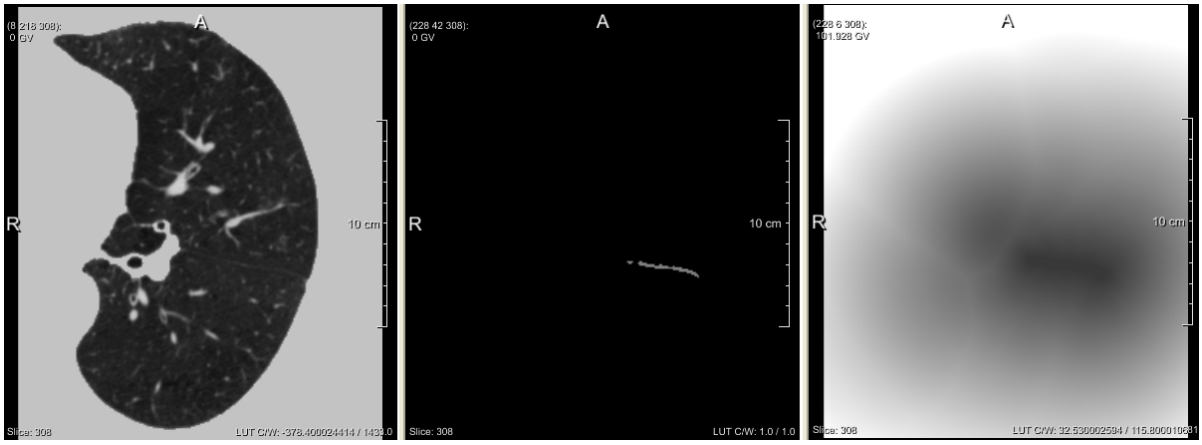


Figure 6.3: Distance map for the fixed image F: (Left) An unseen CT scan image,(Center)Estimated fissure from the unseen image, (Right)Distance map of the fissure

As can be seen in figure 6.4, the intensity values in the $dist_{M_i}$ and the $dist_F$ are quite close to each other at the points where fissure matches in both the images. Therefore, this minima in the difference of the intensities of the corresponding points in $dist_{M_i}$ and $dist_F$ tells us about matching of the fissure. If the intensity difference is higher at a particular location, we know that both the images are dissimilar at that location. Therefore, the extent of difference in the intensity values of these two distance maps tell us about the similarity between the corresponding deformed moving image and the fixed image. We have experimented with two type of weights which are calculated based on these distance maps:

6.1.2 Registration error based weights

I. Isgum and van Ginneken [2009] used absolute difference in intensities between a deformed moving image DM_i and the fixed image F as weight for the corresponding segmentation DS_i. But we compute the weights using $dist_{M_i}$ and $dist_F$:

$$D_i = |dist_{M_i} - dist_F| \tag{6.1}$$

Weight for a voxel x of i^{th} input segmentation DS_i:

$$W_i(x) = \frac{1}{|D_i(x) * g_\sigma(x) + \epsilon|} \tag{6.2}$$

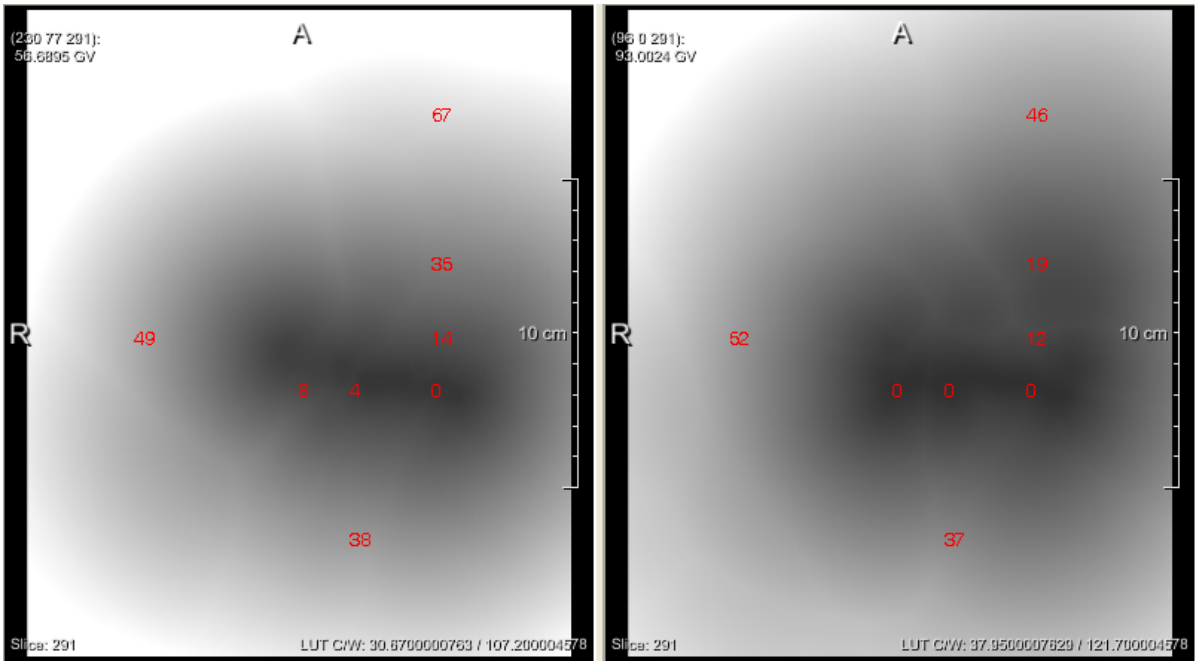


Figure 6.4: (Left) Distance map for a moving image i.e. $dist_{M_i}$. (Right) Distance map for the corresponding fixed image i.e. $dist_F$. The intensity values at various corresponding points are shown. Lower the difference in the intensity values of a corresponding voxel, more is the similarity between the two images.

Since these weights can take any positive value, I. Isgum and van Ginneken [2009] normalized the weight of a voxel x with the sum of weights for the same voxel from other deformed moving images. The normalized weight $W_{i,norm}(x)$ can take any between 0 to 1.

$$W_{i,norm}(x) = \frac{W_i(x)}{\sum_{i=1}^{N} W_i(x)} \tag{6.3}$$

6.1.3 Normalized correlation coefficient (NCC) based weights

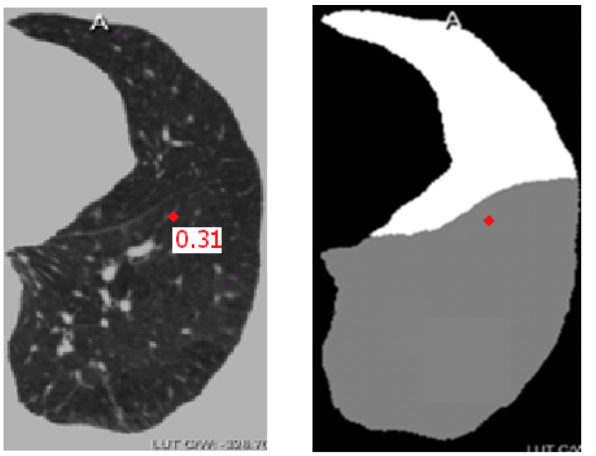
Artaechevarria et al. [2009] used normalized correlation between a deformed moving image DM_i and fixed image F to find weight for an input segmentation DS_i. Instead we use $dist_{M_i}$ and $dist_F$ for computing NCC based weights The intensity values in the rectangular neighborhood (of radius r) of a voxel x in $dist_{M_i}$ are stored in a vector $V_{i,r}(x)$. Similarly, intensity values in the rectangular neighborhood of the voxel x in $dist_F$ are stored in a vector $V_r(x)$. The *local* weight for the voxel x in DS_i is calculated by using expression:

$$W_{i,r}(x) = NCC[V_{i,r}(x), V_r(x)] \tag{6.4}$$

Therefore, we consider intensity values neighborhood of each voxel for finding its local weight. This weight value is always between 0 and 1.

6.2 Importance of Local Confidence Values

Any of the weight generation strategy discussed above provides a *local* weight for each voxel in each input image. See figure 6.5. Although this much information is enough to perform weighted voting. But we aim to investigate if can make a more informed decision than weighted voting. Section 6.5 contains details and example of many such decision strategies which require more information compared to weighted voting.



Deformed Moving Image Deformed Segmentation Image

Figure 6.5: (Left) Weight for the highlighted voxel in a deformed moving image DM_i is 0.31. It represents the weight of corresponding voxel in deformed segmentation DS_i. (Right) Weight for the highlighted voxel, with label 1, is 0.31.

In figure 6.5, we know weight (i.e. 0.31) associated with label-1 for the highlighted voxel. But there is no information on the weight associated with other labels. The knowledge of how remaining 0.69 weight is divided between the remaining label-2 and label-0 can be useful. In this case, on can see that none of the neighborhood voxel has label-0, therefore we expect label-0 to have little or no part of remaining 0.69 weight. We propose to calculate *confidence values* from these weights to find confidence associated with all the labels.

6.3 Confidence Calculation

The registration accuracy based weight $W_i(x)$ for any voxel in DM_i provide us with a weight value for the label of corresponding voxel x in DS_i. We propose to use this weight $W_i(x)$ as basis for computing confidence values associated with all the labels, for voxel x. For confidence calculation, we make use of labels in neighborhood of the voxel x in DS_i. Consider an example given below:

Table 6.1: Example

1	1	2
2	0	0
0	1	1

Therefore, Let us assume:
N_{tot} = Total number of voxels in neighborhood of center voxel with weight W_0.

N_0 = Total number of voxels in neighborhood, with label 0 (center voxel).
N_1 = Number of voxels in neighborhood, with label 1.
N_2 = Number of voxels in neighborhood, with label 2.
Now,

$$\begin{aligned}
&\text{Confidence for label 0} = W_0 + (N_0 - 1) \times \frac{1 - W_0}{N_{tot} - 1} = W_0 + (2 - 1) \times \frac{1 - W_0}{9 - 1} \\
&\text{Confidence for label 1} = (N_1) \times \frac{1 - W_0}{N_{tot} - 1} = (4) \times \frac{1 - W_0}{9 - 1} \\
&\text{Confidence for label 2} = (N_2) \times \frac{1 - W_0}{N_{tot} - 1} = (2) \times \frac{1 - W_0}{9 - 1}
\end{aligned} \tag{6.5}$$

After calculation of confidence values for all the voxels in all the inputs, we can arrange them in a matrix in following manner:

Table 6.2: Example

	Atlas1			Atlas2		
Voxel Number	label0	label1	label2	label0	label1	label2
1.	0.4	0.3	0.3	0.1	0.5	0.4
2.	0.9	0.1	0.0	0.4	0.2	0.4
3.	0.25	0.15	0.6	0.2	0.3	0.5
4.	0.8	0.1	0.1	0.1	0.45	0.45
5.	0.7	0.15	0.15	0.15	0.35	0.5

Let us have a close look at the equation set 6.5. Confidence for label-0 contains two terms: first is the weight which is calculated based on success of registration, second is the factor $(N_0 - 1) \times \frac{1-W_0}{N_{tot}-1}$. This second factor enhances confidence on label-0 if any voxel in the neighborhood also contains label-0. Therefore for a region of uniform intensity, e.g. black background with intensity 0, the overall confidence for label-0 adds up to 1. For such a case confidence on label-2 and label-1 would be 0 since the neighborhood does not contain any voxel with label-2 or label-1, $N_2 = 0$ and $N_1 = 0$.

6.4 Parameters in the Current Setup

6.4.1 Neighborhood radius for finding weights

The σ of gaussian kernel as shown in equation 6.2 and neighborhood radius of rectangular window as shown in equation 6.4 are the parameters used in weight calculation. Both of these parameters ensure smooth variation of weights. Thus, avoid the effect of noise. High value of these parameters can cause excess smoothing which may result into loss of local nature of weights, also it increases computation time of weights. While a very low value may result into noise affected weights. Therefore we need to study this trade-off and find an agreeable value of these parameters for our dataset.

6.4.2 Neighborhood radius for confidence calculation

For confidence calculation, for voxel x, we look at the rectangular neighborhood of the voxel as shown in table 6.2. Increasing the size of this rectangular window may not have any effect

of the confidence values of voxel-As shown in figure 6.6. But it will not be the case for voxel-B since voxels with different labels will also be captured in the larger rectangular window. Therefore confidence value will decrease for label-1 and increase for label-2. Therefore, changing neighborhood radius for confidence calculation may have different effect for voxels at different locations within the same image.

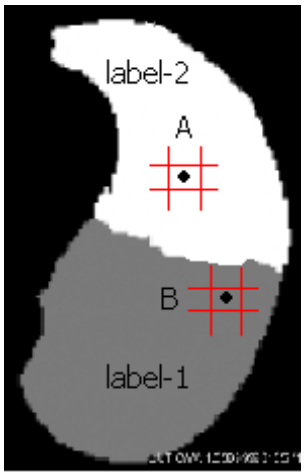


Figure 6.6: Rectangular window for confidence calculations of voxel A and B is shown. Increasing the size of this window will not change confidence values for voxel-A.

6.4.3 Different approaches of confidence calculation

We experimented with two different approaches for confidence calculation: (1) within (2) across. The *across* approach is an improvement over our *within* approach.

Within normalization: The example of confidence calculation given in section 6.3 employs *within* normalization. In this approach N_0, N_1, N_2, and N_{tot} shown in equation 6.5 are calculated within the rectangular window around voxel x in an input segmentation DS_i.

Across normalization: For calculating confidence values for a voxel x in an input image, we consider the neighborhood of of corresponding voxel in all the other input segmentations. See figure 6.7. For confidence calculation of voxel x in input, we compute the total number of voxels with label-0 in the neighborhood of all the inputs which represents N_0. Similarly, N_1, N_2, and N_{tot} are computed by looking at all the inputs instead of only input-1 (figure 6.7). Now, we can use the same equation set 6.5 for finding the confidence values of voxel x.

6.5 Fixed Rules

Artaechevarria et al. [2009] and I. Isgum and van Ginneken [2009] used *weighted voting* for fusing the weights found based on success of registration. Once we have confidence values as shown in table 6.2, we can use many other combining strategies such as summation, product, maximum, median, minimum, voting. These are usually termed as *Fixed Rules*, since they behave in same manner irrespective of input data. Fixed rules are commonly used for *combining pattern classifiers*. Any fixed rule may perform better than other rules depending on input data. Usually, a fixed rule which gives least training error is applied on test data. Therefore, we applied for all the fixed rules mentioned above to investigate which rule works best for our dataset. The

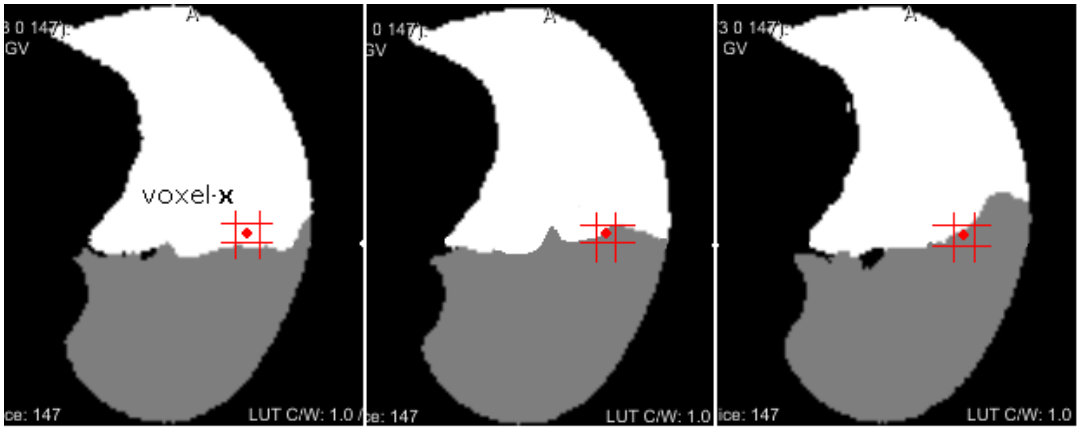


Figure 6.7: For *across* confidence calculation of voxel X in input image-1, we need to look at the corresponding neighborhood of other input images too.

working of summation and product fixed rules is explained with the help of an example shown in figure 6.8.

	$Atlas_1$			$Atlas_2$			SUM Rule			PRODUCT Rule		
Voxel Number	**$Label_0$**	**$Label_1$**	**$Label_2$**	**$Label_0$**	**$Label_1$**	**$Label_2$**	**$Label_0$**	**$Label_1$**	**$Label_2$**	**$Label_0$**	**$Label_1$**	**$Label_2$**
1.	0.4	0.3	0.3	0.1	0.5	0.4	0.5	0.8	0.7	0.04	0.15	0.12
2.	0.9	0.1	0.0	0.4	0.2	0.4	1.3	0.3	0.4	0.36	0.02	0.0
3.	0.25	0.15	0.6	0.2	0.3	0.5	0.45	0.45	1.1	0.05	0.045	0.3
4.	0.8	0.1	0.1	0.1	0.45	0.45	0.9	0.55	0.55	0.08	0.045	0.045
5.	0.7	0.15	0.15	0.15	0.35	0.5	0.85	0.5	0.65	0.105	0.0525	0.075

SUM of confidence values of corresponding labels

PRODUCT of confidence values of corresponding labels

Figure 6.8: A demonstration of SUM and PRODUCT rule is shown. The rules are applied for each voxel independently. In SUM rule, the confidence values of corresponding labels are added and the sum represent overall confidence for that label. Finally a label with highest confidence is selected. The selected labels are highlighted red for both SUM and PRODUCT rule. For voxel number 3, we see a tie between $label_0$ and $label_1$. Such ties are broken randomly.

6.6 Improving Computation Speed for the Algorithm

Even the very bad input segmentations, whose fissure are not correct in any local region as compared to ground truth, posses same label for a majority of voxels. See figure 6.9. For all the voxels where a clear unanimity exists among all the input segmentations, we do not need to make any further computation of weight or confidence value. We simply rely on the unanimous decision for such voxels. So we calculate weight/confidences and apply fixed rules only for those voxels for whom unanimity does not hold. This technique was used in Artaechevarria et al. [2009].

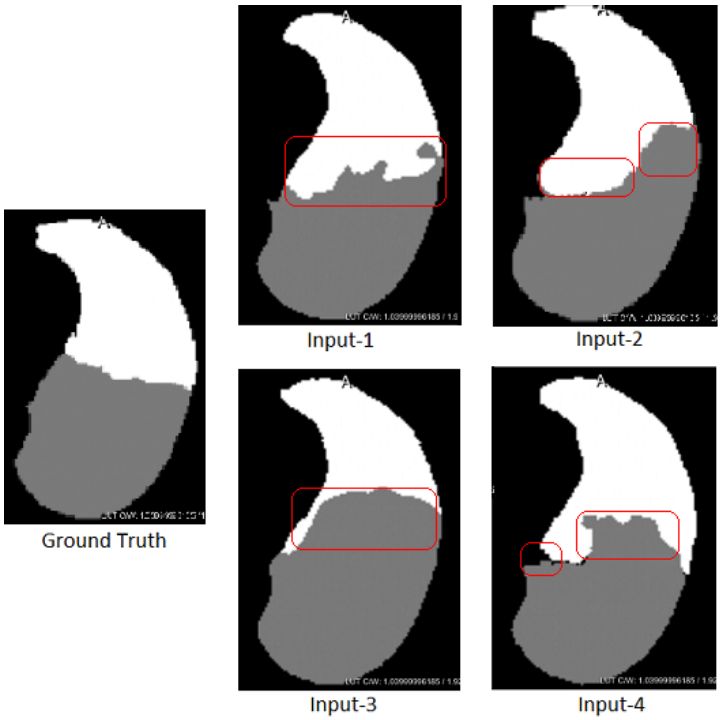


Figure 6.9: A set of four input segmentations and their corresponding ground truth. We can see clear *unanimity* for all the voxels except those which are encircled.

6.7 Results and Discussion

Figure 6.10, 6.11, 6.12, and 6.13 show results of various fixed rules for NCC and registration error based weights, where WVOTE represents *weighted voting* which uses NCC or registration error based weights without any confidence calculation. MAJORITY represents majority voting which is applied directly on the input segmentations and no weights or confidences need to be computed. VOTE represents voting rule applied on the confidence values. As we can see that none of the fixed rules improve the result over majority voting. We experimented with a number of values of σ for gaussian, size of the rectangular window for NCC weights and different approaches of confidence calculation, but still majority voting performs better. A comparison between the results of weighted voting and majority voting helps in understanding the reason for such results.

6.7.1 Comparison between majority and weighted voting

Majority voting performs better than weighted voting, for both the NCC and the registration error based weights. As we discussed in section 2.4.4, weighted voting is expected to improve over majority. The poor performance of weighted voting is due to *bad* weights which do not represent the correctness of individual input segmentations correctly. We propose the idea of confidence calculation to make a more informed decision than weighted voting. Since confidence values make use of these weights, we cannot expect good results from any fixed rule which is applied on these confidence values.
The registration error and NCC weights were used in previous researches and weighted voting performed significantly better than majority for the dataset used in I. Isgum and van Ginneken [2009] and Artaechevarria et al. [2009]. But for our dataset it does not work to the expectation.

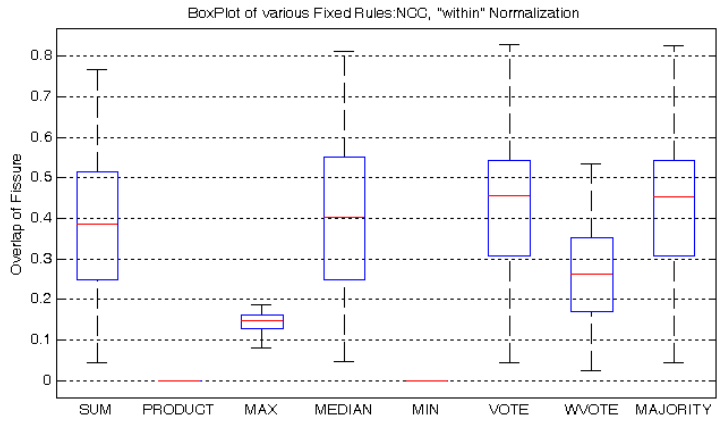


Figure 6.10: Box-plot of different fixed rules for weights based on normalized correlation coefficients and *within* normalization used for confidence calculation. Rectangular window of radius 1 was used for both weight and confidence calculation.

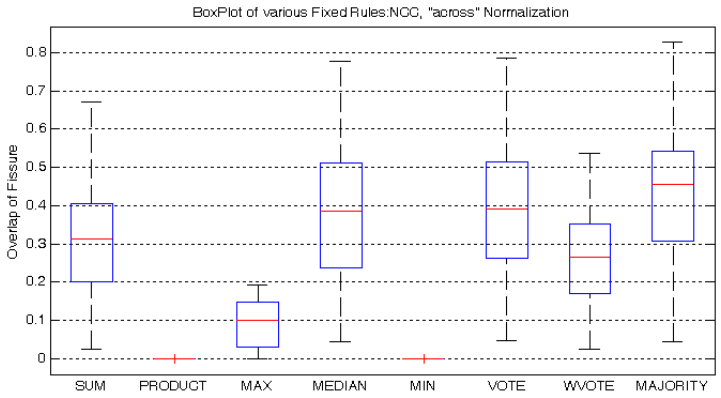


Figure 6.11: Box-plot of different fixed rules for weights based on normalized correlation coefficients and *across* normalization used for confidence calculation. Rectangular window of radius 1 voxel was used for both weight and confidence calculation. WVOTE represents *weighted voting* which makes use of NCC weights without any confidence calculation. MAJORITY represents majority voting which is applied directly on the input segmentations and no weights or confidences need to be computed. VOTE represents voting rule applied on confidence values.

Therefore, one needs to address the problem of finding *reliable* weights based on the success of registration for pulmonary CT scan data. This area is still open to research. We leave it as a message for further research that it would be right approach to check authenticity of weights before developing any strategy which uses these weights. Comparing the performance of weighted voting with majority voting is an easy way to do that. Once we have good weights, it may be the case that some of the fixed rules improve over weighted voting. In our experiment summation, median, and vote fixed rules consistently performed better than weighted voting (figure 6.12, 6.13, 6.10, and 6.11). But we cannot be sure of this in absence of good weights.
Our registration strategy (setup-A ,see section subsubsec:registrationsetupOLD), focuses on aligning initially estimated fissure in the fixed and moving images. As a result of this, the intensity values in the deformed moving image and fixed image are arbitrary and do not convey any information on registration quality. Therefore we do not compute the weights from the fixed and deformed moving images directly. We used distance map images for weight calculation, since

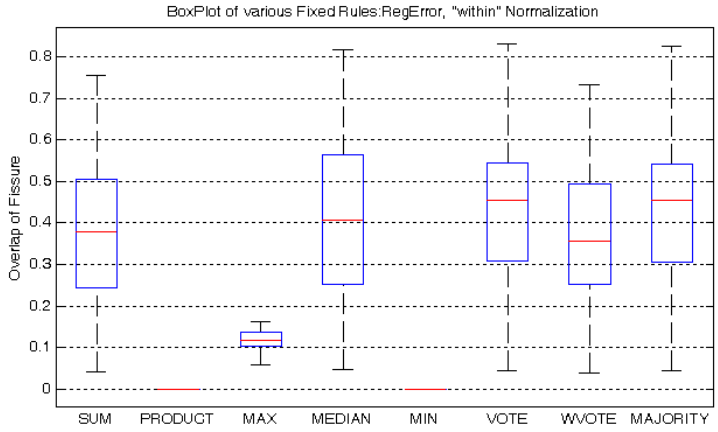


Figure 6.12: Box-plot of different fixed rules for weights based on registration error and *within* normalization used for confidence calculation. Gaussian kernel radius (σ) and rectangular window for confidence calculation both were taken as 1 voxel.

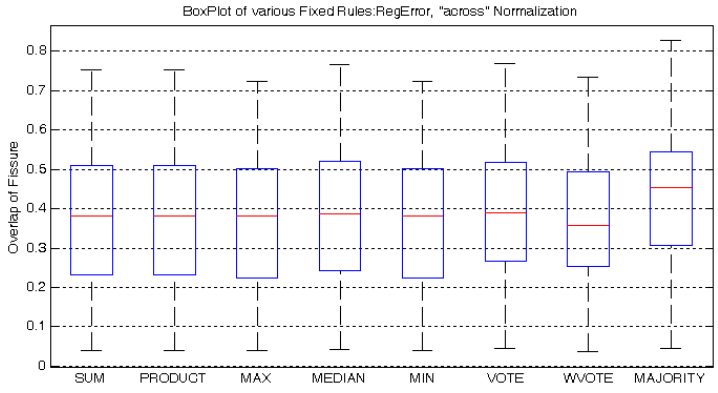
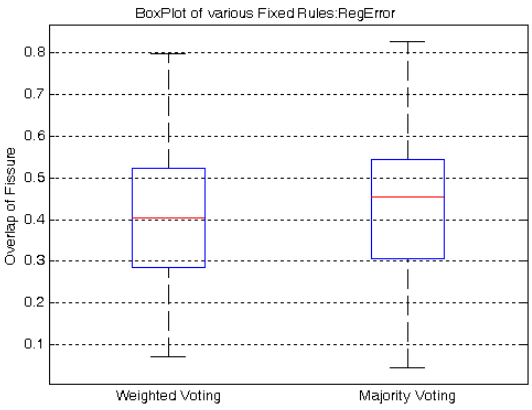


Figure 6.13: Box-plot of different fixed rules for weights based on registration error and *across* normalization used for confidence calculation. Gaussian kernel radius (σ) and rectangular window for confidence calculation both were taken as 1 voxel.

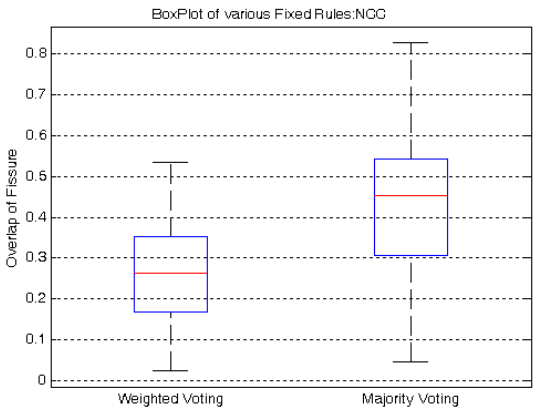
a difference in the intensity of corresponding voxels (in the two distance maps) gives an idea of the accuracy of registration. See section 6.1.1 for details.

6.7.2 Across versus Within confidence calculation

We experimented with two ways of confidence calculation: *across* and *within*. We consider the *across* approach as an improvement over the *within* approach. In the *within* approach, confidence values for a voxel x are calculated within the rectangular neighborhood an input image. If entire neighborhood is wrong as shown in figure 6.15, we get wrong confidence values. In the case shown in the figure 6.15, confidence for label-1 is 1 and 0 for label-2. (since there is no voxel with label-2 within the rectangular window of x, i.e. $N_2 = 0$ in equation 6.5) One look at the ground truth is enough to tell that these confidence values are wrong. Since the corresponding voxel in ground truth is surrounded with voxels of label-2, therefore confidence for label-2 cannot be 0.
In case of *across* confidence calculation, N_2 is calculated by looking at all the input segmentations. There are higher chances that $N_2! = 0$ anymore, thus confidence for label-2 will be non-zero and it will depend on the total number of voxels with label-2 in the corresponding



(a) Weights are based on registration error.



(b) Weights are based on normalized correlation coefficients.

Figure 6.14: Comparison of majority voting with weighted voting. For both the cases radius = 1 voxel is used for weight calculation.

neighborhoods of all the inputs. See figure 6.15.

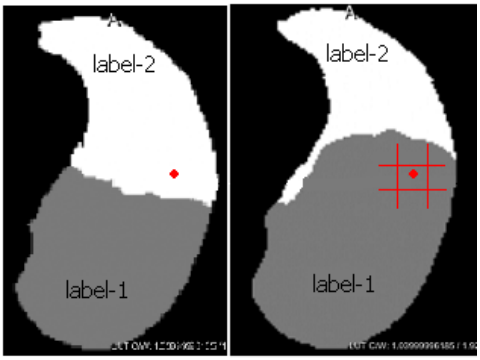


Figure 6.15: Input segmentation has wrong label for the voxel shown.

6.7.3 Some other ideas which did not work well

I found a 1000 different ways not to make a light bulb before making it once and for all- Thomas Edison. This section contains some of the ideas which did not work well but they may serve as motivation for future research. We have only discussed the ideas and results are not shown. Once we realized that we first need to find good weights for our dataset, we methods of finding weight using distance map of fissure of fixed and deformed moving image. Read section 6.1.1 before going through this section.

Using signed distance map Distance map that we used for weight generation has lowest value (0) at the fissure and further, it increases on both sides of this oblique fissure (figure 6.4). But many deformed moving images encounter the case shown in figure 6.16 where mismatch of fissure should cause completely different intensity values for voxels close to fissure. But for the voxel highlighted in figure 6.16, the distance maps from fixed and deformed moving images posses same value. Similarly for many other nearby voxels intensity difference in these two distance maps may be quite low which results into high weights (NCC or registration error). Now

the problem arises because this voxel has label-1 in the corresponding deformed segmentation, therefore this high weight value is assigned to label-1. But if we look at the segmentation of fixed image, this voxel should have label-2. So here we are assigning a high weight value to a wrong label. The same story goes for most of the neighbor voxels. We must not forget that these wrongly weighted voxels are close to fissure and have important role in deciding the final estimation of fissure obtained through fixed rule strategy.

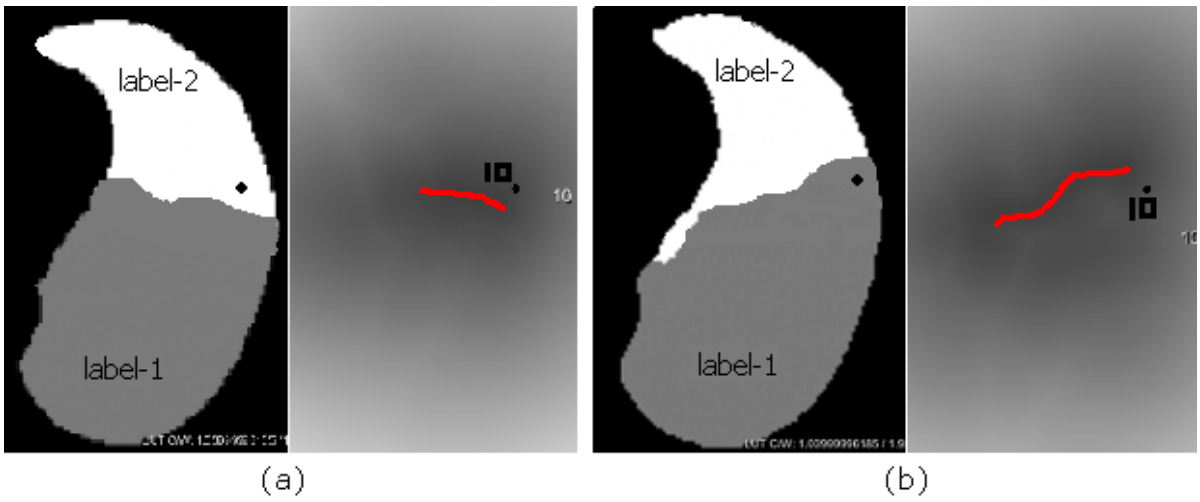


Figure 6.16:

In order to solve this problem, we decided to use distance maps with positive values on one side of fissure and negative values on the other side. In this case, the voxel shown in figure 6.16 will have intensity value 10 in distance map for deformed moving image and intensity value 10 in the corresponding fixed image distance map. As a result, the mismatch of fissure causes higher difference in intensity values and so lower weight for the voxel.
Since intensity value are increasing on both sides of fissure, we used gradient of distance map in Y-direction, which resulted into positive values above fissure and negative below. The sign of each voxel of this gradient image can be multiplied with the corresponding intensity values in distance map to obtain the signed distance map that we discussed above.
As we already know this idea did not work since the problem lies in the fissure which is used to compute distance map. Consider the case shown in figure 6.17. The areas enclosed within the red rectangle has positive high values in the distance map for fixed image while the similar red rectangle contains quite low positive values (and zeros) in the distance map for moving image.This arbitrary difference in the intensity values causes unreliable weights for the region within the red rectangle.
Using fissure mask: Once we realized that having incomplete fissure creates such a distance map which causes false weights in the region away from fissure, we decided to focus only on the region close to initially detected fissure of fixed image. We prepared a mask from this fissure. See figure 6.18. Furthermore, we calculate weights, using distance map, only for voxels within the mask. For remaining voxel we directly calculate confidence values, considering weight to be 0, based on the label of neighborhood voxels (refer equation 6.5). This did not work either and we are not sure about the cause.

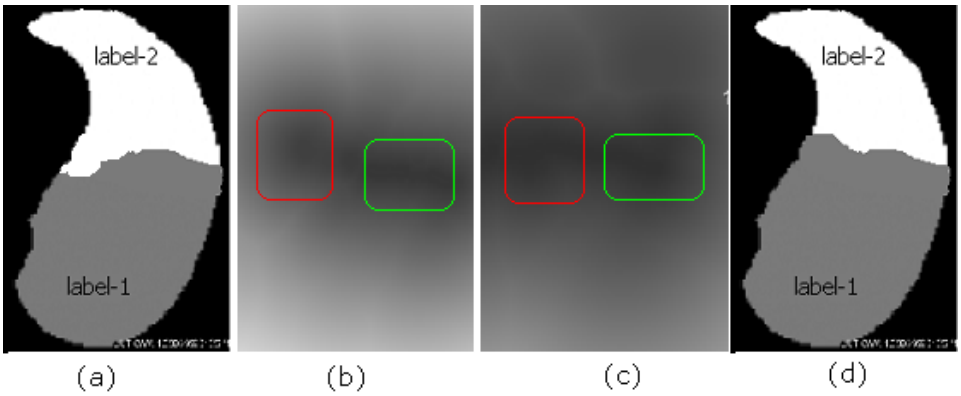


Figure 6.17: (a) Ground Truth, (b) Distance map of initially detected fissure in a fixed image, (c) Distance map of fissure present in one of the deformed moving images, (d) Deformed segmentation corresponding to (c). The red rectangle shown in (b) contains positive high values while the similar rectangle contains several 0 in (c). This mismatch in values causes wrong weights for the region enclosed within the red rectangle. While it is not the case for voxels enclosed within green rectangle. Both the distance maps contains zeros or low positive values which eventually generates high weights for this region. Since the deformed segmentation contains fissure correctly for this region, we expect weights to be high.

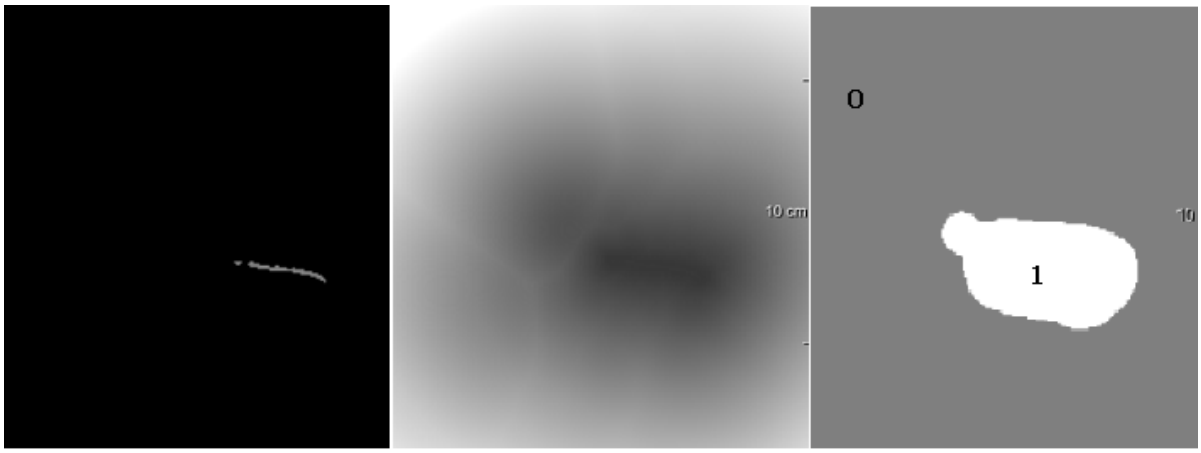


Figure 6.18: (Left) Initially detected fissure from a fixed image, (center) distance map of the fissure, (right) mask around this fissure. We rely on the values of signed distance map only within the mask.

6.8 Conclusion

The chapter is focussed on two aspects of the fusion of segmentations fusion: (1) Finding reliable weights based on the success of individual atlas based segmentations, (2) Suitable fusion strategy for those weights. We discussed several fusion strategies or *Fixed Rules* as is usually called. Any of these fixed rules (sum, median, product, maximum, and minimum) may perform better than others depending on the input data. Therefore we experimented with all of these rules to see which one works best for our pulmonary CT scan dataset.
In previous publications, few researchers have computed weights based on the success of registration and performed weighted voting for fusing the weights. As is evident from section 6.2 and 6.5, fixed rules are more informed ways of decision making as compared to weighted voting.

We propose the idea of *confidence values* which uses:

1. Weights based on the success of individual atlas based segmentation, i.e., success of registration.

2. Labels of voxels in the neighborhood of voxel in consideration.

We further investigated two methods of confidence calculation: *Within*, which makes use of the labels present in the neighborhood of the voxel in consideration. It considers the neighborhood within the atlas which contains this voxel. While *Across* confidence calculation considers the corresponding neighborhood of all the atlases. See section 6.3 for details. This way we attempt to extract more information from the input data. We do not only look at the weights obtained from registration but also at the labels of segmentations to be fused. This way equation for confidence calculation depends on the reliability of weights also. As we found in our experiment, NCC and registration error based weights do not give reliable information about the quality of registration for our pulmonary dataset. The difficulty in aligning vessels and fissures from one patient data to other makes it impossible to calculation registration quality based weight directly from the fixed and the deformed moving image. Therefore, we used distance map of fixed and moving images for weight calculation. For the fixed image, the distance map is calculated by using its initially estimated fissure, which was also used to support the registration in setup-A (3.3.4.1). In case of moving image, we make use fissure present in the deformed moving image for calculating distance map. See section 6.1.1 for details on distance map computation. As a key finding of our experiment, the weights generated from these two distance maps are not reliable which can simply be observed if we compare the performance of weighted voting with majority voting as shown in section 6.7.1. We experimented several ways to find reliable weights using these distance maps but unfortunately none of them succeeded. Therefore, finding good weights, based on success of registration, for pulmonary CT scan data is still open to research. We leave it as a message for future research that one should first compare the performance of weighted voting with majority voting and if weighted voting shows significant improvement over majority, then we can assume such weights to be reliable. In absence of good weights, we cannot comment on the results obtained from applying fixed rules on confidence values. The only reason of presenting these results is to show that they do not provide any improvement.

Chapter 7

Ideas Which Did Not Work

In this chapter some of the ideas which did not work as we expected are elaborated.

7.1 Local STAPLE Algorithm

STAPLE makes use of global weights which are based on confusion matrix. This confusion matrix is computed between the input segmentations and the estimate of ground truth. We tried developing *local non-overlapping cube-wise STAPLE* and *local overlapping cube-wise STAPLE*. As the name suggests, weights were computed using a non-overlapping and overlapping sliding window respectively. Figure 7.1 shows an output image obtained from local overlapping STAPLE on using the binary left lung segmentations as input.

Figure 7.1: A sample output from local overlapping cube-wise STAPLE algorithm on using cube-size 50 .

We are not able to explain the exact reason for such result. One possibility is that confusion matrix calculation within a local window was not reliable. It may be because a window may not contain the same number of labels as the entire input image. Therefore confusion matrix computation within such a window may be erroneous. Our initial guess is that use of prior information for each label may improve these results. But it requires more experiments to say something further on it.

7.2 SIMPLE Fixed Rules

In chapter 4 we saw that DICE overlap between an input segmentation the estimate of ground truth provides reliable weights for the segmentation. For the same reason the performance of weighted voting in SIMPLE was found to be significantly better than majority voting. In the next chapter 6, we studied about confidence calculation and fixed rules that can be applied on those confidence values. The major problem with our confidence values based approach was the unavailability of reliable weights. Since registration error based weights were not found to be reliable. Therefore we decided to use DICE overlap based weights as basis for confidence calculation. This also gives as another opportunity to see if any fixed rule can outperform the weighted voting.

We combined the SIMPLE algorithm with our confidence calculation and fixed rule approach. The *SIMPLE fixed rule* algorithm starts with computing an initial estimate of ground truth. Similar to *local overlapping cube-wise SIMPLE algorithm*, DICE value for each overlapping cube within an input segmentation is comptued. Now we use these DICE weights to compute the confidence values as shown in equation 6.5. Then average and standard deviation of the confidence values is used to find a threshold value corresponding to each label present in the input segmentations. The confidence values less than the corresponding thresholds are made 0, i.e., those voxels are rejected. Finally, fixed rules are applied on the remaining confidence values.

Problem with the algorithm: The overlap of 3 voxel wide fissure obtained from this algorithm was worse compared to even majority voting. The problem lies in using DICE overlap as weights for computing confidence values for each voxel in the input segmentations. As we mentioned in the chapter 4, DICE overlap based weights are not reliable for small cube sizes. Consider a case shown in figure on using the cube-size 50 for DICE calculation. Since a number of voxels with label 1 match in both the images, the dice overlap for the voxel in the center of the window shown comes out to be 0.65. This voxel's confidence value for label 1 would go even higher than 0.65 depending on the label of neighbor voxels. This is the reason for which DICE based weights cannot be used for confidence calculation. A large cube size gives reliable DICE weights, but a large number of voxels may have same label in the input segmentation and the estimate of ground truth which gives usually high DICE weight. Confidence value will add more to this already high weight. Therefore label 1 gets a very high confidence value for this center voxel while we can see in the figure 7.2 that the center voxel has label 2 in the estimate of ground truth. Need of higher cube size is need for reliable DICE weight and the larger cube causes false confidence values for the voxel in the center of the window.

For computation of confidence values we need a separate weight value for each voxel. DICE overlap give provides kind of a global weight value. Therefore it is not a good idea to use DICE values for confidence calculation.

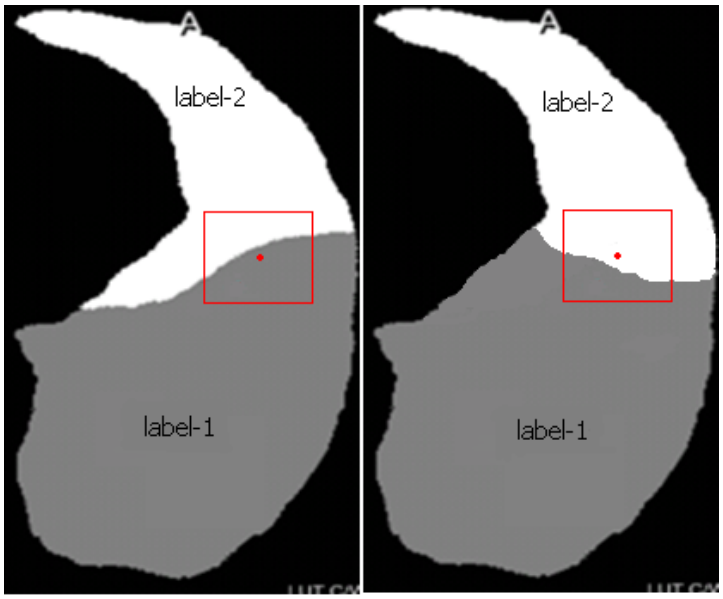


Figure 7.2: SIMPLE fixed rule: fissure in the segmentation of the moving image does not match with the estimate of ground truth, within the enclosed cube of size 50. But since a number of label-1 and label-2 voxels match in both the images, therefore DICE overlap for say label-1 comes out to be 0.65. Now using 0.65 as weight for confidence calculation increases the final confidence value for label 1 even more, depending on the number of neighbors with label 1.

Chapter 8

Discussion and Conclusion

8.1 Discussion

In the current work, we have introduced several segmentation fusion approaches for coping with the local errors in the individual segmentations. These individual segmentations are obtained as a result of multi-atlas based segmentation, applied on the *inter-patient* pulmonary CT data. The local registration errors in the individual ABSs cause the locally erroneous segmentations. The accuracy of a segmentation is different from the other, therefore it is of key importance to find the weights of segmentation images. Since the input segmentations for our dataset are locally correct, we research on various methods of finding the local weights for input segmentations. Once these local weights are found, the second task would be the fusion of these weights, which can be done by several fusion strategies such as weighted voting or fixed rules. We have experimented with three type of approaches: First, which computes the local weights by comparing an input segmentation with an estimate of the ground truth. Second, which computes the local weights based on the local success of the registration. Third is a hybrid approach of the first two.
In the first type of approach, we have proposed several modifications in the SIMPLE algorithm. The SIMPLE algorithm uses global weights of input segmentations and weighted voting is used as the fusion strategy. Some of the modifications worked well for our dataset. We incorporated these features in the SIMPLE: re-inclusion of data, local weight calculation, increasing sensitivity (boundary), and multi-label version. Out of these four features, *local* and *multi-label* versions show significant improvement over the SIMPLE, for our dataset.
If we find the input set to be locally correct in different regions, which is the case with our dataset, then local SIMPLE algorithms can significantly improve over SIMPLE. Local non-overlapping cube wise (LNOC) algorithm is computationally less expensive as compared to the local overlapping cube-wise (LOC) algorithm. But LNOC shows improvement over SIMPLE for a specific cube size 100 only which may be because of the lesser boundary effect for such high cube-size. The reason for change in the performance with cube-size is caused by three factors: (1) Smaller cube-size provides more local weights and as we increase the cube-size, the weights become more and more global. Therefore, we lose the advantage of local fusion strategy. (2) As shown in the figure 4.13, we may encounter jumps in the fissure at intersection-surfaces of the two cubes. A small cube-size may cause many intersection-surfaces within a fissure and consequently may cause many jumps. Therefore, a large cube size may provide more smooth fissure. (3) A very small cube-size may not contain enough voxels to compute DICE measure reliably. It is difficult to say how these 3 factors will interact with each other. Therefore it

is difficult to say beforehand that which cube-size may work better for a given set of atlases. We propose the LOC algorithm as an improvement over LNOC. The LOC algorithm shows statistically significant improvement over SIMPLE for cube-size greater than 35 (table 4.3). We believe that the reason for such result is because DICE overlap cannot be computed reliably for smaller cube-sizes (less than 35). One point must be noted that LOC algorithm is devoid of boundary-effect, which may contribute to the improved performance.

In our dataset, a segmentation image of the right lung contains 4 labels (0, 1, 2, and 3). Therefore, the *multi-label* version of SIMPLE is important for lobular segmentation of the right lung. As we already noticed, in the case of binary inputs overlapping cube-wise algorithm works best for multi-label data as well. The results of multi-label overlapping cube-wise algorithm provide significant improvement over SIMPLE (cube-size ¿ 25), thus over other two multi-label fusion algorithms too, such as majority voting and STAPLE. Since SIMPLE already outperforms these two algorithms.

$\alpha = 1/f$ works well for all of our good performing SIMPLE algorithms such as binary and multi-label local overlapping cube-wise and global multi-label. f-value denotes fraction of selected segmentations in previous iteration. The same set of α values was used for both: the left and the right lung. Also, these α values are optimum for both the registration setups: A and B. Therefore, we conclude that choice of α is specific to an algorithm and $\alpha = 1/f$ is a good choice for all the good performing algorithms. Appendix 9.4 contains the optimum *alpha* values.

If all the input segmentations are wrong at a particular local region, then most probably their fusion output would also be wrong at that location. The only way to improve this situation is to obtain better input segmentations. We experimented with registration setup-B with the same motivation. Quite to our surprise the registration improves the input segmentation to great extent. All the inputs are close to the manual segmentation. We consider the right lung as a more difficult data to register as compared to left lung, since alignment of both the horizontal and the oblique fissure is required by image registration. But we again see almost perfect registration similar to the case of the left lung. It would not be wrong to say that the problem of lobe segmentation is resolved to the accuracy of manual segmentation. We find the registration setup-A useful, since inaccuracy of input segmentations makes it a difficult segmentation fusion problem. We need such an input set to test out various algorithms. Each algorithm performs almost equally good on using inputs generated from setup-B, therefore it is not possible to do comparative study of algorithms for such an ideal input set. The important observation in this experiment is role of the success of registration for improving the output of multi-atlas based image segmentation.

In the second type of approach, which computes weights directly based on the local success of registration, we proposed to use distance maps for the fixed and the moving images to compute the local weights for individual segmentations. The distance map for the fixed image is generated using its initially estimated fissure. We make use of the deformed moving image for computing the other distance map. We compute registration error and normalized correlation coefficient (NCC) based weights from these two distance maps. These weights can be used to apply weighted voting fusion strategy. We further propose to compute the confidence values based on these weights. These confidence values make use of not only accuracy of registration based weights, but also use the labels present within a pre-defined neighborhood in the input segmentations. Since these confidence values employ more information than weighted voting, we wanted to test if this extra information can help in improving the fusion output. But as it turns out, the weights computed using the distance maps do not provide reliable estimate of the local accuracy of the input segmentations. Therefore, the confidence values computed using these

weights also do not give correct results. The computation of weights based on the local success of registration is still open to research for the pulmonary CT data. Since, these registration error and NCC based weights have reported to improve the performance of weighted voting over majority voting, in some other application areas. Although these weights are directly computed from the fixed image and the deformed moving image in the literature. Finally, we believe that if one can compute the registration based weights reliably, the application of fixed rules on the confidence values (computed based on these weights) may improve the results over the weighted voting.
In the third hybrid approach, we use the local cube-wise DICE overlap based weights as basis for computing the confidence values. Since we found the DICE overlap based weights reliable for our dataset, we aimed to test if the application of confidence values and fixed rules can improve the performance over local cube-wise SIMPLE which employs weighted voting. In this approach, the selection (or rejection) of data is done using the threshold computed by the mean and standard deviation of the corresponding confidence values. But this approach did not outperform even majority voting. The problem is associated with the need of a cube size higher than 25 for reliable DICE calculation. See figure 7.2 for details. The confidence values expect per voxel based local weights which may take any value between 1 to 0. But DICE overlap of a cube size 35 (or greater) usually has higher overlap because of the matching of greater part within the cube even if there are local errors within this cube. Computing confidence of an already high weight value makes it even higher depending on the label of the neighborhood voxels (equation 6.5). Eventually these confidence values convey wrong information about the local accuracy of input segmentations since confidence is usually high because of high weight values. Refer section 7.2 for details.

8.2 Conclusion

In conclusion, the proposed local overlapping cube-wise SIMPLE algorithm outperforms SIMPLE for binary segmentation by considering local weights for selecting the locally good regions within a segmentation. The multi-label version of local overlapping cube-wise algorithm outperforms the other multi-label segmentation approaches such as majority voting and STAPLE. Thus the local fusion approach performs better than global approaches. The registration setup-B, tends to the accuracy of the manual segmentation for lobe detection.

8.3 Future Work

The following topics deserve to be subject of further study:

1. Finding reliable local weights based on the local registration accuracy.

2. Focusing on a hybrid approach which uses not only input segmentations but also local registration errors for finding the reliable confidence values. Studying performance of fixed rules on the confidence values would be interesting. One can also use *trainable classifiers* (support vector machine, linear discriminant classifier, k-nearest neighbor etc.) instead of fixed rules.

3. In our local-overlapping cube-wise approach, a fixed cube size is used. It would be interesting to see if a variable cube size provide any improvement. For such an approach, the cube size should be selected based on the anatomical information of an atlas. E.g. entire region near *hilum* can be selected within a single cube, since hilum region is difficult to register and using many small cubes only increases the computation and may be the *boundary-effect* too, while we can consider using many smaller cubes for the region near fissure in order to obtain more local weights.

4. Local STAPLE, which makes use of the local weights based on the confusion matrix computed within a predefined window size.

Chapter 9

APPENDIX

9.1 Parameters for Registration Setup-A

9.1.1 Affine transformation

//********** Image Types (FixedInternalImagePixelType "float")
(FixedImageDimension 3)
(MovingInternalImagePixelType "float")
(MovingImageDimension 3)
//********** Components (Registration "MultiResolutionRegistration")
(FixedImagePyramid "FixedRecursiveImagePyramid")
(MovingImagePyramid "MovingRecursiveImagePyramid")
(Interpolator "BSplineInterpolator")
(Metric "AdvancedNormalizedCorrelation")
(Optimizer "AdaptiveStochasticGradientDescent")
(ResampleInterpolator "FinalBSplineInterpolator")
(Resampler "DefaultResampler")
(Transform "AffineTransform")

//********** Pyramid // Total number of resolutions (NumberOfResolutions 5)
(ImagePyramidSchedule 16 16 16 8 8 8 4 4 4 2 2 2 1 1 1)

//********** Transform (AutomaticScalesEstimation "true")
(AutomaticTransformInitialization "true")
(HowToCombineTransforms "Compose")

//********** Optimizer
// Maximum number of iterations in each resolution level:
(MaximumNumberOfIterations 1000)
(AutomaticParameterEstimation "true")
(UseAdaptiveStepSizes "true")

//********** Metric
//********** Several (WriteTransformParametersEachIteration "false")

(WriteTransformParametersEachResolution "true")
(WriteResultImageAfterEachResolution "false")
(WriteResultImage "false")
(ShowExactMetricValue "false")
(ErodeMask "false")
(UseDirectionCosines "true")

//********** ImageSampler
//Number of spatial samples used to compute the mutual information in each resolution level:
(ImageSampler "RandomCoordinate")
(NumberOfSpatialSamples 2000)
(NewSamplesEveryIteration "true")
(UseRandomSampleRegion "false")
(MaximumNumberOfSamplingAttempts 5)

//********** Interpolator and Resampler
//Order of B-Spline interpolation used in each resolution level:
(BSplineInterpolationOrder 1)
//Order of B-Spline interpolation used for applying the final deformation:
(FinalBSplineInterpolationOrder 3)
//Default pixel value for pixels that come from outside the picture:
(DefaultPixelValue 0)

9.1.2 First B-spline transformation

//********** Image Types
(FixedInternalImagePixelType "float")
(FixedImageDimension 3)
(MovingInternalImagePixelType "float")
(MovingImageDimension 3)

// ********** Components
(Registration "MultiResolutionRegistration")
(FixedImagePyramid "FixedRecursiveImagePyramid")
(MovingImagePyramid "MovingRecursiveImagePyramid")
(Interpolator "BSplineInterpolator")
(Metric "AdvancedNormalizedCorrelation")
(Optimizer "AdaptiveStochasticGradientDescent")
(ResampleInterpolator "FinalBSplineInterpolator")
(Resampler "DefaultResampler")
(Transform "BSplineTransform")

// ********** Pyramid
// Total number of resolutions
(NumberOfResolutions 5)
(ImagePyramidSchedule 16 16 16 8 8 8 4 4 4 2 2 2 1 1 1)

```
// ********** Transform
(FinalGridSpacingInPhysicalUnits 10.0 10.0 10.0)
(GridSpacingSchedule 8.0 8.0 4.0 2.0 1.0)
(HowToCombineTransforms "Compose")

// ********** Optimizer
// Maximum number of iterations in each resolution level:
(MaximumNumberOfIterations 1000)
(AutomaticParameterEstimation "true")
(UseAdaptiveStepSizes "true")

// ********** Metric
// Just using the default values for the NC metric

// ********** Several

(WriteTransformParametersEachIteration "false")
(WriteTransformParametersEachResolution "true")
(WriteResultImageAfterEachResolution "false")
(WritePyramidImagesAfterEachResolution "false")
(WriteResultImage "false")
(ShowExactMetricValue "false")
(ErodeMask "false")
(UseDirectionCosines "true")

// ********** ImageSampler
//Number of spatial samples used to compute the mutual information in each resolution level:
(ImageSampler "RandomCoordinate")
(NumberOfSpatialSamples 2000)
(NewSamplesEveryIteration "true")
(UseRandomSampleRegion "false")
(SampleRegionSize 50.0 50.0 50.0)
(MaximumNumberOfSamplingAttempts 50)

// ********** Interpolator and Resampler
//Order of B-Spline interpolation used in each resolution level:
(BSplineInterpolationOrder 1)
//Order of B-Spline interpolation used for applying the final deformation:
(FinalBSplineInterpolationOrder 3)
//Default pixel value for pixels that come from outside the picture:
(DefaultPixelValue 0)
```

9.1.3 Second B-spline transformation

```
// ********** Image Types
(FixedInternalImagePixelType "float")
(FixedImageDimension 3)
(MovingInternalImagePixelType "float")
(MovingImageDimension 3)

// ********** Components
(Registration "MultiResolutionRegistration")
(FixedImagePyramid "FixedRecursiveImagePyramid")
(MovingImagePyramid "MovingRecursiveImagePyramid")
(Interpolator "BSplineInterpolator")
(Metric "AdvancedNormalizedCorrelation")
(Optimizer "AdaptiveStochasticGradientDescent")
(ResampleInterpolator "FinalBSplineInterpolator")
(Resampler "DefaultResampler")
(Transform "BSplineTransform")

// ********** Pyramid
// Total number of resolutions
(NumberOfResolutions 5)
(ImagePyramidSchedule 4 4 4 3 3 3 2 2 2 1 1 1 1 1 1)

// ********** Transform
(FinalGridSpacingInPhysicalUnits 5.0 5.0 5.0)
(GridSpacingSchedule 16.0 8.0 4.0 2.0 1.0)
(HowToCombineTransforms "Compose")

// ********** Optimizer
// Maximum number of iterations in each resolution level:
(MaximumNumberOfIterations 2000)
(AutomaticParameterEstimation "true")
(UseAdaptiveStepSizes "true")

// ********** Metric
// Just using the default values for the NC metric

// ********** Several
(WriteTransformParametersEachIteration "false")
(WriteTransformParametersEachResolution "true")
(WriteResultImageAfterEachResolution "false")
(WritePyramidImagesAfterEachResolution "false")
(WriteResultImage "true")
(ShowExactMetricValue "false")
(ErodeMask "false" "false" "true" "true" "true")
(UseDirectionCosines "true")
```

// ********** ImageSampler
//Number of spatial samples used to compute the mutual information in each resolution level:
(ImageSampler "RandomCoordinate")
(NumberOfSpatialSamples 2000)
(NewSamplesEveryIteration "true")
(UseRandomSampleRegion "false")
(SampleRegionSize 50.0 50.0 50.0)
(MaximumNumberOfSamplingAttempts 50)

// ********** Interpolator and Resampler
//Order of B-Spline interpolation used in each resolution level:
(BSplineInterpolationOrder 1)
//Order of B-Spline interpolation used for applying the final deformation:
(FinalBSplineInterpolationOrder 3)
//Default pixel value for pixels that come from outside the picture:
(DefaultPixelValue 0)

9.2 Parameters for Registration Setup-B

9.2.1 Affine transformation

// ********** Image Types
(FixedInternalImagePixelType "float")
(FixedImageDimension 3)
(MovingInternalImagePixelType "float")
(MovingImageDimension 3)

// ********** Components
(Registration "MultiResolutionRegistration")
(FixedImagePyramid "FixedRecursiveImagePyramid")
(MovingImagePyramid "MovingRecursiveImagePyramid")
(Interpolator "BSplineInterpolator")
(Metric "AdvancedNormalizedCorrelation")
(Optimizer "AdaptiveStochasticGradientDescent") (ResampleInterpolator "FinalBSplineInterpolator")
(Resampler "DefaultResampler")
(Transform "AffineTransform")

// ********** Pyramid
// Total number of resolutions
(NumberOfResolutions 5)
(ImagePyramidSchedule 16 16 16 8 8 8 4 4 4 2 2 2 1 1 1)

// ********** Transform

```
(AutomaticScalesEstimation "true")
(AutomaticTransformInitialization "true")
(AutomaticTransformInitializationMethod "GeometricalCenter")
(HowToCombineTransforms "Compose")

// ********** Optimizer
// Maximum number of iterations in each resolution level:
(MaximumNumberOfIterations 1000)
(AutomaticParameterEstimation "true")
(UseAdaptiveStepSizes "true")

// ********** Metric

// ********** Several
(WriteTransformParametersEachIteration "false")
(WriteTransformParametersEachResolution "true")
(WriteResultImageAfterEachResolution "false")
(WriteResultImage "true")
(ResultImageFormat "mha")
(ShowExactMetricValue "false")
(ErodeMask "false")
(UseDirectionCosines "true")

// ********** ImageSampler
//Number of spatial samples used to compute the mutual information in each resolution level:
(ImageSampler "RandomCoordinate")
(NumberOfSpatialSamples 2000)
(NewSamplesEveryIteration "true")
(UseRandomSampleRegion "false")
(MaximumNumberOfSamplingAttempts 5)

// ********** Interpolator and Resampler
//Order of B-Spline interpolation used in each resolution level:
(BSplineInterpolationOrder 1)
//Order of B-Spline interpolation used for applying the final deformation:
(FinalBSplineInterpolationOrder 3)
//Default pixel value for pixels that come from outside the picture:
(DefaultPixelValue 0)
```

9.2.2 B-spline transformation

```
// ********** Image Types
(FixedInternalImagePixelType "float")
(FixedImageDimension 3)
(MovingInternalImagePixelType "float")
(MovingImageDimension 3)
```

```
// ********** Components
(Registration "MultiMetricMultiResolutionRegistration")
(FixedImagePyramid "FixedRecursiveImagePyramid" "FixedRecursiveImagePyramid")
(MovingImagePyramid "MovingRecursiveImagePyramid" "MovingRecursiveImagePyramid")
(Interpolator "BSplineInterpolator" "BSplineInterpolator")
(Metric "AdvancedMeanSquares" "AdvancedMeanSquares")
(Optimizer "AdaptiveStochasticGradientDescent")
(ResampleInterpolator "FinalBSplineInterpolator")
(Resampler "DefaultResampler")
(Transform "BSplineTransform")

// ********** Pyramid
// Total number of resolutions
(NumberOfResolutions 5)
(ImagePyramidSchedule 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1)

// ********** Transform
(FinalGridSpacingInPhysicalUnits 10.0 10.0 10.0)
(GridSpacingSchedule 8.0 8.0 4.0 2.0 1.0)
(HowToCombineTransforms "Compose")

// ********** Optimizer
// Maximum number of iterations in each resolution level:
(MaximumNumberOfIterations 1000)
(AutomaticParameterEstimation "true")
(UseAdaptiveStepSizes "true")

// ********** Metric
// Just using the default values for the NC metric
(Metric0Weight 1.0)
(Metric1Weight 4.0)

// ********** Several
(WriteTransformParametersEachIteration "false")
(WriteTransformParametersEachResolution "true")
(WriteResultImageAfterEachResolution "false")
(WritePyramidImagesAfterEachResolution "false")
(WriteResultImage "true")
(ResultImageFormat "mha")
(ShowExactMetricValue "false")
(ErodeMask "false")
(UseDirectionCosines "true")

// ********** ImageSampler
```

```
//Number of spatial samples used to compute the mutual information in each resolution level:
(ImageSampler "MultiInputRandomCoordinate" "MultiInputRandomCoordinate")
(NumberOfSpatialSamples 2000)
(NewSamplesEveryIteration "true")
(UseRandomSampleRegion "false")
(SampleRegionSize 50.0 50.0 50.0)
(MaximumNumberOfSamplingAttempts 50)

// ********** Interpolator and Resampler
//Order of B-Spline interpolation used in each resolution level:
(BSplineInterpolationOrder 1)
//Order of B-Spline interpolation used for applying the final deformation:
(FinalBSplineInterpolationOrder 3)
//Default pixel value for pixels that come from outside the picture:
(DefaultPixelValue 0)
```

9.3 Pseudo Codes of Algorithms

9.3.1 Local Slice-Wise SIMPLE Algorithm

Algorithm 3 Local Slice-Wise SIMPLE

Require: DS_i, α

initialize $iter = 0$
perform majority voting of DS_i to obtain an initial estimate of ground truth, i.e., $E(GT)_0$
count the number of slice, $numSlices$ in an input image DS_i
for $j = 0$ to $numSlices$ **do**
  initialize $NumberOfSlices_{0,j} = numSlices$
end for
while Any slice from any input is rejected in last iteration **do**
  for $j = 0$ to $numSlices$ **do**
    for $i = 1$ to N **do**
      compute $Dice_{i,j}$ = overlap of j^{th} slice from DS_i and $E(GT)_{iter}$ for all the slices that are not rejected.
    end for
    compute $Mean_{iter,j}(Dice_{:,j})$ and $StdDev_{iter,j}(Dice_{:,j})$ using only the slices that are not rejected.
    compute $Threshold_{iter,j} = Mean_{iter,j} - \alpha \times StdDev_{iter,j}$
  end for
  for $i = 0$ to N **do**
    for $j = 0$ to $numSlices$ **do**
      if $Dice_{i,j} < Threshold_{iter,j}$ **then**
        reject j^{th} slice from DS_i
      end if
    end for
  end for
  for $j = 0$ to $numSlices$ **do**
    count $NumberOfRejectedSlices_{iter,j}$
    $NumberOfSlices_{iter+1,j} = NumberOfSlices_{iter,j} - NumberOfRejectedSlices_{iter,j}$
    perform weighted voting of remaining $DS_{i,j}$ to obtain $E(GT)_{iter+1,j}$
  end for
  $iter = iter + 1$
end while
$E(GT)_{iter}$ is final fusion result, i.e., $E(GT)$

9.3.2 Local Non-Overlapping Cube-Wise SIMPLE Algorithm

Algorithm 4 Local Non-Overlapping Cube-Wise SIMPLE

Require: $DS_i, \alpha, cubeSize$

initialize $iter = 0$
perform majority voting of DS_i to obtain an initial estimate of ground truth, i.e., $E(GT)_0$
Each DS_i is divided into non-overlapping cubes of size $cubeSize$
count the number of cubes, $numCubes$ in an input image DS_i
for $j = 0$ to $numCubes$ **do**
  initialize $NumberOfCubes_{0,j} = numCubes$
end for
while Any cube from any input is rejected in last iteration **do**
  for $j = 0$ to $numCubes$ **do**
    for $i = 1$ to N **do**
      compute $Dice_{i,j}$ = overlap of j^{th} cube from DS_i and $E(GT)_{iter}$ for all the cubes that are not rejected.
    end for
    compute $Mean_{iter,j}(Dice_{:,j})$ and $StdDev_{iter,j}(Dice_{:,j})$ using only the cubes that are not rejected.
    compute $Threshold_{iter,j} = Mean_{iter,j} - \alpha \times StdDev_{iter,j}$
  end for
  for $i = 0$ to N **do**
    for $j = 0$ to $numCubes$ **do**
      if $DICE_{i,j} < Threshold_{iter,j}$ **then**
        reject j^{th} cube from DS_i
      end if
    end for
  end for
  for $j = 0$ to $numCubes$ **do**
    count $NumberOfRejectedCubes_{iter,j}$
    $NumberOfCubes_{iter+1,j} = NumberOfCubes_{iter,j} - NumberOfRejectedCubes_{iter,j}$
    perform weighted voting of remaining $DS_{i,j}$ to obtain $E(GT)_{iter+1,j}$
  end for
  $iter = iter + 1$
end while
$E(GT)_{iter}$ is final fusion result, i.e., $E(GT)$

9.3.3 Boundary SIMPLE Algorithm

Algorithm 5 Increasing Sensitivity (Boundary) SIMPLE

Require: DS_i, α

perform majority voting of DS_i to obtain an initial estimate of ground truth, i.e., $E(GT)_0$
initialize $NumberOfInputs_0 = NumberOfInputs$
initialize $iter = 0$
while $NumberOfInputs_{iter} \neq NumberOfInputs_{iter-1}$ **do**
  for $i = 0$ to N **do**
    compute $Dice_i$ = overlap of $3D$ **boundary** of DS_i and $E(GT)_{iter}$ for all the input segmentations that are not rejected.
  end for
  compute $Mean_{iter}(Dice)$ and $StdDev_{iter}(Dice)$ using only the input segmentations that are not rejected.
  compute $Threshold_{iter} = Mean_{iter} - \alpha \times StdDev_{iter}$
  for $i = 0$ to N **do**
    if $Dice_i < Threshold_{iter}$ **then**
      reject DS_i
    end if
  end for
  count $NumberOfRejectedInputs_{iter}$
  $NumberOfInputs_{iter+1} = NumberOfInputs_{iter} - NumberOfRejectedInputs_{iter}$
  $iter = iter + 1$
  perform weighted voting of remaining DS_i to obtain $E(GT)_{iter}$. $DICE_i$ is used as weight for DS_i.
end while
$E(GT)_{iter}$ is final fusion result, i.e., $E(GT)$

9.3.4 Global Multi-Label SIMPLE Algorithm

Algorithm 6 Global Multi-Label SIMPLE algorithm

Require: DS_i, α

perform majority voting of DS_i to obtain an initial estimate of ground truth, i.e., $E(GT)_0$
initialize $NumberOfInputs_0 = NumberOfInputs$
computer the number of labels in an input image $NumLabels$
initialize $iter = 0$
while $NumberOfInputs_{iter} \neq NumberOfInputs_{iter-1}$ **do**
  for $l = 0$ to $NumLabels - 1$ **do**
    for $i = 1$ to N **do**
      compute $Dice_{i,l}$ = overlap of voxels with l_{th} label in DS_i and $E(GT)_{iter}$ for all the input segmentations that are not rejected.
    end for
    compute $Mean_{iter,l}(Dice_{:,l})$ and $StdDev_{iter,l}(Dice_{:,l})$ using only the input segmentations that are not rejected.
    compute $Threshold_{iter,l} = Mean_{iter,l} - \alpha \times StdDev_{iter,l}$
  end for
  for $i = 0$ to N **do**
    if $Dice_i < Threshold_{iter}$ **then**
      reject DS_i
    end if
  end for
  count $NumberOfRejectedInputs_{iter}$
  $NumberOfInputs_{iter+1} = NumberOfInputs_{iter} - NumberOfRejectedInputs_{iter}$
  $iter = iter + 1$
  perform weighted voting of remaining DS_i to obtain $E(GT)_{iter}$. $DICE_i$ is used as weight for DS_i.
end while
$E(GT)_{iter}$ is final fusion result, i.e., $E(GT)$

9.4 α Values for SIMPLE Algorithms

Table 9.1: Binary segmentation: Set of α values used for various algorithms

α Index	re-inclusion	boundary	slicewise	slicewise boundary
α_0	-0.35	0.5	0.75	1.75
α_1	-0.25	0.75	1.0	2.0
α_2	-0.15	1.0	1.25	2.25
α_3	-0.05	$0.25/f$	1.5	2.5
α_4	0	$0.5/f$	$\sqrt{(1-f)}$	2.75
α_5	0.25	$0.75/f$	$1/f$	$1/f$

Table 9.2: Binary segmentation: Set of α values used for various algorithms

α Index	LNOC	LNOCB	LOC	LNOCR
α_0	0.75	0.75	1.0	0.0
α_1	1.0	1.0	1.25	0.25
α_2	1.25	1.25	1.5	0.5
α_3	1.5	1.5	1.75	$0.25/f$
α_4	$1/f$	$\exp(-f)$	$1/f$	$0.1/f$
α_5	$\exp(-f)$	$1/f$	$0.5/f$	$0.01/f$

Table 9.3: Multi-Label: Set of α values used for various algorithms

α Index	Global	LOC	LNOC
α_0	1.0	1.0	0.75
α_1	1.25	1.25	1.25
α_2	$0.25/f$	1.5	$0.25/f$
α_3	$0.5/f$	1.75	$0.5/f$
α_4	$0.75/f$	$0.5/f$	$0.75/f$
α_5	$1.0/f$	$1/f$	$1/f$

9.5 Boxplots for SIMPLE Algorithms

9.5.1 Multi-label versus binary SIMPLE

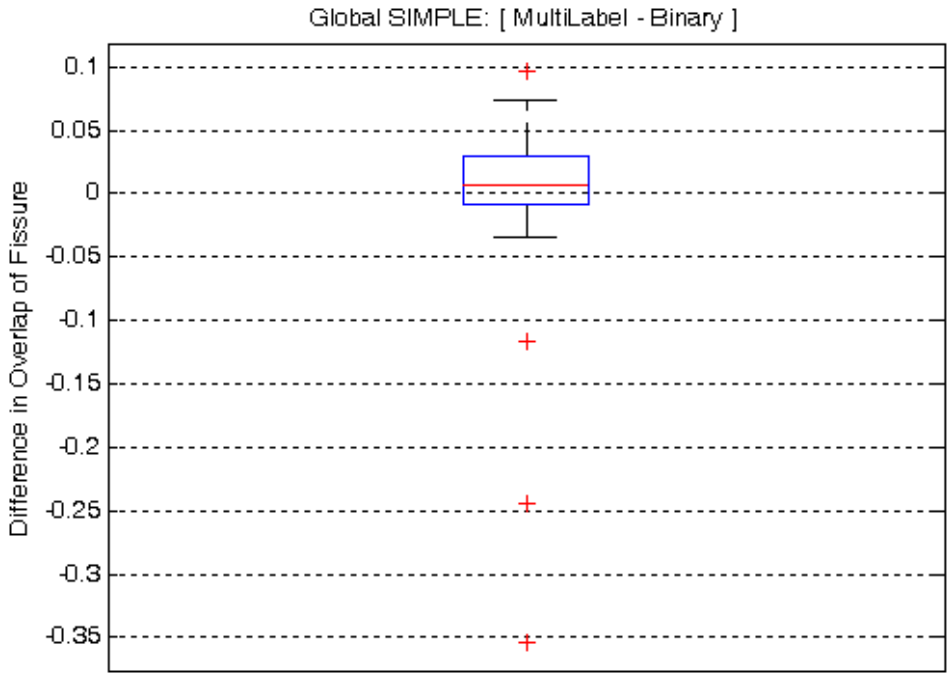


Figure 9.1: A comparison of multilabel and binary global SIMPLE.

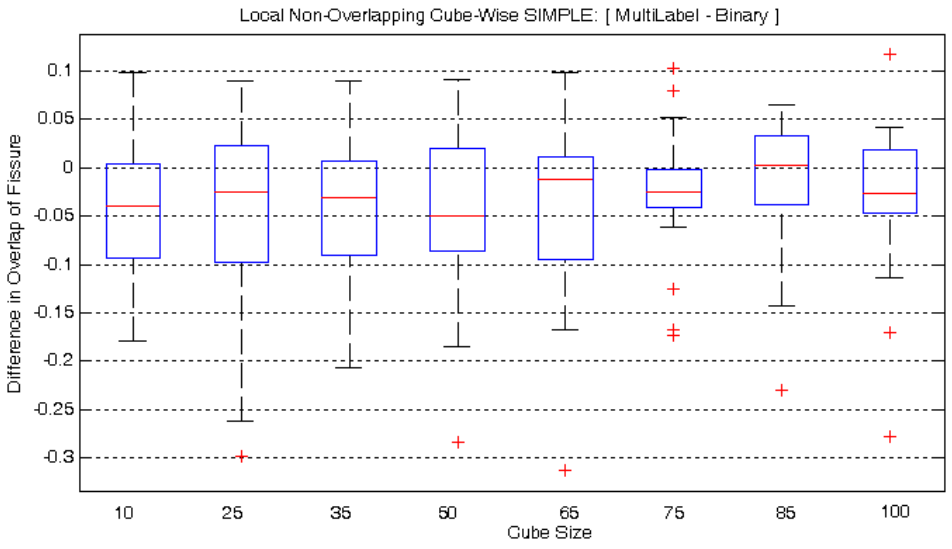


Figure 9.2: A comparison of multilabel and binary non-overlapping cube-wise SIMPLE.

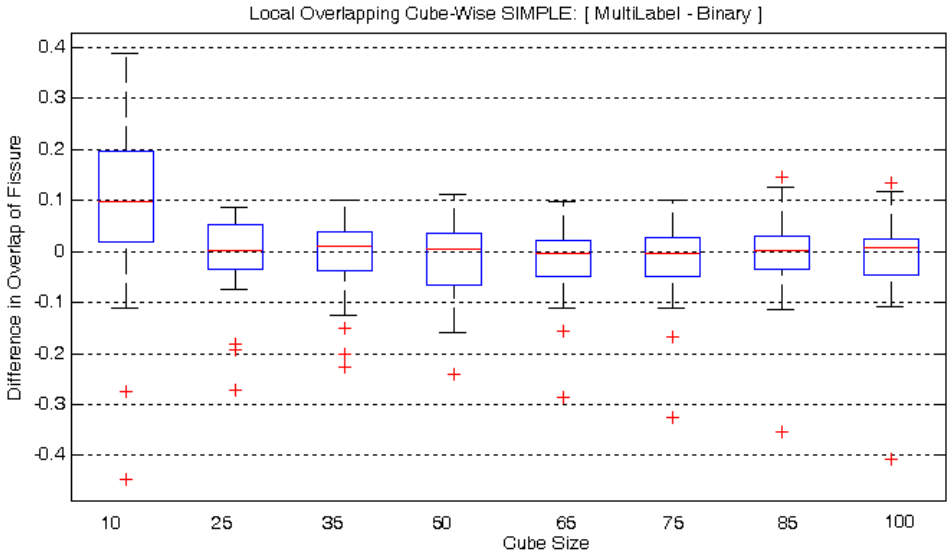


Figure 9.3: A comparison of multilabel and binary overlapping cube-wise SIMPLE.

Bibliography

A randomized trial comparing lung-volumereduction surgery with medical therapy for severe emphysema. *New England Journal of Medicine*, 348(21):2059–2073, 2003. doi: 10.1056/NEJMoa030287.

P. Aljabar, R.A. Heckemann, A. Hammers, J.V. Hajnal, and D. Rueckert. Multi-atlas based segmentation of brain images: Atlas selection and its effect on accuracy. *NeuroImage*, 46(3):726 – 738, 2009. ISSN 1053-8119. doi: DOI:10.1016/j.neuroimage.2009.02.018. URL http://www.sciencedirect.com/science/article/B6WNP-4VP4WC5-2/2/0c9e225b2baf87400dbc24d5135517e1.

X. Artaechevarria, A. Munoz-Barrutia, and C. Ortiz de Solorzano. Combination strategies in multi-atlas image segmentation: Application to brain MR data. *Medical Imaging, IEEE Transactions on*, 28(8):1266–1277, August 2009.

O. Carmichael, H. Aizenstein, S. W. Davis, J. Becker, P. M. Thompson, C. Meltzer, and Yanxi Liu. Atlas-based hippocampus segmentation in alzheimer's disease and mild cognitive impairment. *NeuroImage*, 27(4):979 – 990, 2005.

A. Guimond, J. Meunier, and J.P. Thirion. Average brain models: A convergence study. 77(2): 192–210, February 2000.

Rolf A. A. Heckemann, Joseph V. V. Hajnal, Paul Aljabar, Daniel Rueckert, and Alexander Hammers. Automatic anatomical brain MRI segmentation combining label propagation and decision fusion. *NeuroImage*, 33(1):115–126, October 2006. ISSN 1053-8119. doi: 10.1016/j.neuroimage.2006.05.061. URL http://dx.doi.org/10.1016/j.neuroimage.2006.05.061.

D. L. G. Hill, P. G. Batchelor, M. Holden, and D. J. Hawkes. Medical image registration. 46 (3):R1 – R45, 2001.

A. Rutten M. Prokop M. A. Viergever I. Isgum, M. Staring and B. van Ginneken. Multi-atlas-based segmentation with local decision fusion-application to cardiac and aortic segmentation in ct scans. *Medical Imaging, IEEE Transactions on*, 28(7):1000–1010, 2009.

L. Ibáñez, W. Schroeder, L. Ng, and J. Cates. *The ITK Software Guide*. Kitware, Inc. ISBN 1-930934-15-7, second edition, 2005.

Josef Kittler, Ieee Computer Society, Mohamad Hatef, Robert P. W. Duin, and Jiri Matas. On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20: 226–239, 1998.

Arno Klein, Brett Mensh, Satrajit Ghosh, Jason Tourville, and Joy Hirsch. Mindboggle: Automated brain labeling with multiple atlases. *BMC Medical Imaging*, 5(1):7, 2005. ISSN

1471-2342. doi: 10.1186/1471-2342-5-7. URL http://www.biomedcentral.com/1471-2342/5/7.

S. Klein, M. Staring, and J. P. W. Pluim. Evaluation of Optimization Methods for Nonrigid Medical Image Registration Using Mutual Information and B-Splines. *IEEE Transactions on Image Processing*, 16:2879–2890, December 2007. doi: 10.1109/TIP.2007.909412.

S. Klein, U.A. van der Heide, I.M. Lips, M. van Vulpen, M. Staring, and J. P. W. Pluim. Automatic segmentation of the prostate in 3d mr images by atlas matching using localized mutual information. *Medical Physics*, 35(4):1407–1417, April 2008.

Stefan Klein and Marius Staring. elastix. http://elastix.isi.uu.nl, 2004. URL http://elastix.isi.uu.nl.

Robin Langerak, Uulke A. van der Heide, Alexis N. T. J. Kotte, Max A. Viergever, Marco van Vulpen, and Josien P. W. Pluim. Label fusion in atlas-based segmentation using a selective and iterative method for performance level estimation (simple). *IEEE Trans. Med. Imaging*, 29(12):2000–2008, 2010.

H. Lester and S. R. Arridge. A survey of hierarchical non-linear medical image registration. 32 (1):129 – 149, 1999.

B. Li, G. E. Christensen, G. McLennan, E. A. Hoffman, and J. M. Reinhardt. Establishing a normative atlas of the human lung: Inter-subject warping and registration of volumetric CT. *Acad. Radiol.*, 10(3):255, 2003.

Maria Lorenzo-Valds, Gerardo I. Sanchez-Ortiz, Andrew G. Elkington, Raad H. Mohiaddin, and Daniel Rueckert. Segmentation of 4d cardiac MR images using a probabilistic atlas and the EM algorithm. *Medical Image Analysis*, 8(3):255 – 265, 2004. ISSN 1361-8415. doi: DOI:10.1016/j.media.2004.06.005. URL http://www.sciencedirect.com/science/article/B6W6Y-4CVV50R-5/2/539f3691e37256a59aed75626d5eb4d2. Medical Image Computing and Computer-Assisted Intervention - MICCAI 2003.

J. B. A. Maintz and M. A. Viergever. A survey of medical image registration. 2(1):1 – 36, 1998.

H. Park, P. Bland, and C. Meyer. Construction of an abdominal probabilistic atlas and its application in segmentation. *IEEE Trans. Med. Imag.*, 22(4):483 – 492, April 2003.

Torsten Rohlfing and Calvin R. Maurer, Jr. Shape-based averaging. *IEEE Transactions on Image Processing*, 16(1):153–161, Jan. 2007. doi: 10.1109/TIP.2006.884936.

Torsten Rohlfing, Robert Brandt, Calvin R. Maurer, Jr., and Randolf Menzel. Bee brains, B-splines and computational democracy: Generating an average shape atlas. In Lawrence Staib, editor, *IEEE Workshop on Mathematical Methods in Biomedical Image Analysis*, pages 187–194, Kauai, HI, 2001. IEEE Computer Society, Los Alamitos, CA. ISBN 0-7695-1336-0. doi: 10.1109/MMBIA.2001.991733.

Torsten Rohlfing, Robert Brandt, Randolf Menzel, and Calvin R. Maurer, Jr. Evaluation of atlas selection strategies for atlas-based image segmentation with application to confocal microscopy images of bee brains. *NeuroImage*, 21(4):1428–1442, Apr. 2004a. doi: 10.1016/j.neuroimage.2003.11.010. URL http://www.sciencedirect.com/science/article/B6WNP-4BS0F4N-1/2/3452c4a5336f3edc4eb47d0e98f88163.

Torsten Rohlfing, Daniel B. Russakoff, and Calvin R. Maurer, Jr. Performance-based classifier combination in atlas-based image segmentation using expectation-maximization parameter estimation. *IEEE Transactions on Medical Imaging*, 23(8):983–994, Aug. 2004b.

Torsten Rohlfing, Robert Brandt, Randolf Menzel, Daniel B. Russakoff, and Calvin R. Maurer, Jr. Quo vadis, atlas-based segmentation? In Jasjit Suri, David L. Wilson, and Swamy Laxminarayan, editors, *The Handbook of Medical Image Analysis – Volume III: Registration Models*, chapter 11, pages 435–486. Kluwer Academic / Plenum Publishers, New York, NY, Aug. 2005.

Michal Sdika. Combining atlas based segmentation and intensity classification with nearest neighbor transform and accuracy weighted vote. *Medical Image Analysis*, 14(2):219 – 226, 2010. ISSN 1361-8415. doi: DOI:10.1016/j.media.2009.12.004. URL http://www.sciencedirect.com/science/article/B6W6Y-4XY4GT3-1/2/3e0a707c0748b20e44726c11bd39c1e0.

I. Sluimer, M. Prokop, and B. van Ginneken. Towards automated segmentation of the pathological lung in CT), journal = IEEE Trans. Med. Imag., volume = 24, month = August, year = 2005, number = 8, pages = 1025 - 1038,.

J. Stancanello, P. Romanelli, N. Modugno, P. Cerveri, G. Ferrigno, F. Uggeri, and G. Cantore. Atlas-based identification of targets for functional radiosurgery. *Medical Physics*, 33(6):1603 – 1611, 2006.

M. Staring, S. Klein, J.H.C. Reiber, W.J. Niessen, and B.C. Stoel. Pulmonary Image Registration With elastix Using a Standard Intensity-Based Algorithm. In Bram van Ginneken, Keelin Murphy, Tobias Heimann, Vladimir Pekar, Xiang Deng, editor, *Medical Image Analysis for the Clinic: A Grand Challenge, Workshop Proceedings of MICCAI*, volume of *Lecture Notes in Computer Science*, pages – , Beijing, China, September 2010.

C. Svarer, K. Madsen, S. G. Hasselbalch, L. H. Pinborg, S. Haugbol, V. G. Frokjaer, S. Holm, O. B. Paulson, and G. M. Knudsen. MR-based automatic delineation of volumes of interest in human brain PET images using probability maps. *Neuroimage*, 24(4):969–979, February 2005. ISSN 1053-8119. doi: 10.1016/j.neuroimage.2004.10.017. URL http://dx.doi.org/10.1016/j.neuroimage.2004.10.017.

E.M. van Rikxoort, M. Prokop, B. de Hoop, M.A. Viergever, J.P.W. Pluim, and B. van Ginneken. Automatic segmentation of pulmonary lobes robust against incomplete fissures. *IEEE Transactions on Medical Imaging*, 29(6):1286–1296, 2010a.

Eva M. van Rikxoort, Ivana Isgum, Yulia Arzhaeva, Marius Staring, Stefan Klein, Max A. Viergever, Josien P.W. Pluim, and Bram van Ginneken. Adaptive local multi-atlas segmentation: Application to the heart and the caudate nucleus. *Medical Image Analysis*, 14(1):39 – 49, 2010b. ISSN 1361-8415. doi: DOI:10.1016/j.media.2009.10.001. URL http://www.sciencedirect.com/science/article/B6W6Y-4XFGJJN-1/2/b107e3dd298ef173d8b096ed67916578.

Simon K. Warfield, Kelly H. Zou, William M. Wells, and William M. Simultaneous truth and performance level estimation (staple): An algorithm for the validation of image segmentation. *IEEE Trans. Med. Imag*, 23:903–921, 2004.

M. Wu, O. Carmichael, P. Lopez-Garcia, C. S. Carter, and H. J. Aizenstein. Quantitative comparison of air, spm, and the fully deformable model for atlas-based segmentation of functional and structural mrimages. *Human Brain Mapp*, 27(9):747 – 754, 2006.

Changyan Xiao, Marius Staring, Denis Shamonin, Johan H.C. Reiber, Jan Stolk, and Berend C. Stoel. A strain energy filter for 3d vessel enhancement with application to pulmonary ct images. *Medical Image Analysis*, 15(1):112 – 124, 2011. ISSN 1361-8415.

L. Zhang, E.A. Hoffman, and J.M. Reinhardt. Atlas-driven lung lobe segmentation in volumetric x-ray ct images. 25(1):1–16, January 2006.

B. Zitová and J. Flusser. Image registration methods: a survey. 21(11):977–1000, 2003.