RECURSIVE SHORTEST SPANNING TREE ALGORITHMS FOR IMAGE SEGMENTATION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

NESLİHAN YALÇIN BAYRAMOĞLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE

IN

THE DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

JULY 2005

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Canan ÖZGEN
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. İsmet ERKMEN
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Cüneyt F. BAZLAMAÇCI
Supervisor

Examining Committee Members

Prof. Dr. Semih BİLGEN (METU, EE) _____

Asst. Prof. Dr. Cüneyt F. BAZLAMAÇCI (METU, EE) _____

Assoc. Prof. Dr. A. Aydın ALATAN (METU, EE) _____

Assoc. Prof. Dr. Gözde BOZDAĞI AKAR (METU, EE) _____

Ersin ESEN, M.Sc. (TÜBİTAK) _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Neslihan YALÇIN BAYRAMOĞLU

Signature : 

ABSTRACT

RECURSIVE SHORTEST SPANNING TREE ALGORITHMS FOR IMAGE SEGMENTATION

YALÇIN BAYRAMOĞLU, Neslihan
M.Sc., Department of Electrical and Electronics Engineering
Supervisor : Asst. Prof. Dr. Cüneyt F. BAZLAMAÇCI

July 2005, 118 pages

Image segmentation has an important role in image processing because it is a tool to obtain higher level object descriptions for further processing. In some applications such as large image databases or video image sequence segmentations, the speed of the segmentation algorithm may become a drawback of the application. This thesis work is a study to improve the run-time performance of a well-known segmentation algorithm, namely the Recursive Shortest Spanning Tree (RSST). Both the original and the fast RSST found in the literature are analyzed and a comparison is made between these techniques. Simple modifications and an alternative link cost structure are proposed and

evaluated. Finally, a distributed implementation based on a simple image partitioning strategy is attempted. The thesis presents the results of an extensive computational study with respect to both run-time performance and image segmentation quality.

Keywords: Recursive Shortest Spanning Tree, Graph Theory, Image Segmentation.

ÖZ

ÖZYİNELEMELİ EN KISA UZANIM AĞAÇ ALGORİTMALARI İLE GÖRÜNTÜ PARÇALAMA

YALÇIN BAYRAMOĞLU, Neslihan
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği
Tez Yönetcisi : Asst. Prof. Dr. Cüneyt F. BAZLAMAÇCI

Temmuz 2005, 118 sayfa

Görüntü parçalamanın görüntü işlemede önemli bir yeri vardır. Çünkü görüntü parçalama görüntüde bulunan nesleri başka işlemlerde kullanmak üzere daha üst düzey nesne tanımlamasında kullanılan bir yöntemdir. Parçalama yönteminin çalışma hızı büyük veritabanları ya da video görüntü dizileri gibi uygulamalarda kullanıldığında sorun olabilir. Bu tezde görüntü parçalamada kaliteli sonuçlar veren ve oldukça iyi bilinen Özyinelemeli En Kısa Uzanım Ağaç (RSST) algoritması çalışılmıştır. Literatürde yer alan hem orjinal RSST

hem de hızlı RSST incelenmiş ve bu teknikler karşılaştırılmıştır. Basit değişiklikler ve alternatif bağıntı fonksiyonu önerilip değerlendirilmiştir. Son olarak basit bir görüntü bölme stratejisine dayanan dağınık algoritma uygulaması denenmiştir. Bu tez, çalışma zamanı performansının ve parçalanmış resim kalitesinin kapsamlı ölçüm çalışmalarını sunmaktadır.

Anahtar Kelimeler: Özyinelemeli en kısa uzanım ağacı, RSST, çizge kuramı görüntü parçalama.

To my Family

For their Love & Support

ACKNOWLEDGEMENTS

I would like to express my thanks to my supervisor Asst. Prof. Dr. Cüneyt F. Bazlamaçcı for his guidance and support during the preparation of this study.

I also would like to thank to Assoc. Prof. Dr. A. Aydın Alatan for his valuable suggestions and comments.

I wish to express my deepest thanks to my family for their support and understanding.

Finally, I wish to express my gratitude to my husband M. Fatih Bayramoğlu for his precious support, assistance and unattainable understanding at every stage of this study.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

Graph theory has an important application area in computer vision problems and graph theory concepts, algorithms and results are widely used in this field. It is becoming more and more popular to use graph theory in computer vision problems because of its powerful representation capability and ease of use. The importance and status of graph theory in vision problems are considered in [1], which emphasizes the advantages of formulating computer vision problems as graph problems and geometric methods. Graph theory allows vision problems to be represented as pure and abstract structures in which strong accumulation can be accessed. Computer science and operations research areas which developed graph theory are good resources for vision problems. Due to the geometric characteristics of the vision problems, graph theory is proven to be relevant to this field. Algorithms coming from graph theory such as maximum flow, minimum spanning tree, shortest path, maximal common subtree/subgraph, etc. have applications in vision problems. A number of basic techniques that were

designed in the graph algorithms community have recently been applied to many computer vision problems and one such problem is image segmentation [1].

Dividing an image into semantically meaningful several regions by grouping pixels is called segmentation. The main objective in image segmentation is to distinguish objects from the background. Segmentation has an important role in image recognition. The segmentation part of the recognition system gives the decision of the "objects" in the image for further processing, which can be either further description or recognition. In short, segmentation means classification of the pixels of the image to one of the image parts [2].

Graph theory has introduced several methods including normalized cuts, minimum cuts, dominant set, minimum spanning trees, region adjacency graphs, partition trees etc. for image segmentation. Morris *et al.* [4,5] applied graph theory to image segmentation and edge detection. They applied shortest spanning tree to partition the graph and to obtain a segmentation or edge detection recursively. Recursion is used to obtain global information. The recursive shortest spanning tree (RSST) was proven to be highly accurate to define regions [6]. Since it has applications in more complex settings such as video coding, the run-time performance has a great importance. Hence, efficiency became a major concern in later studies. For example, Kwok and Constantinides [7] proposed Fast RSST (FRSST) algorithm. The algorithm is faster than the conventional RSST but segmented images differ slightly. Kwok and Constantinides [8] also proposed a parallel RSST algorithm. It is claimed that this parallel implementation proposal is cost optimal such that the complexity of the RSST algorithm is reduced from $O(n^2)$ to $O(n)$ in the worst case where n is the total number of vertices in the image graph [8].

This study aims to improve the run-time performance of the Recursive Shortest Spanning Tree (RSST) algorithm. Both the original and the fast RSST found in the literature are analyzed and a comparison is made between these

techniques. Simple modifications and an alternative link cost structure are proposed and evaluated. Finally, a distributed implementation based on a simple image partitioning strategy is attempted. The thesis presents the results of an extensive computational study with respect to both run-time performance and image segmentation quality.

The organization of the study is as follows: Chapter 2 describes the problem, namely image segmentation and its relation to graph theory.

Chapter 3 is a summary of the graph theoretic approaches used in image analysis. A literature survey, which concentrates mostly on image segmentation, is presented.

In Chapter 4, Conventional RSST and Fast RSST are examined and some simple modifications are proposed. In these methods, the link weight costs are used as the absolute pixel intensity differences but Chapter 4 also proposes the use of an alternative link weight cost, named as the optimum cost RSST. Finally, a distributed RSST algorithm is developed in this chapter. The implementation details of the discussed methods are presented in Chapter 5.

Chapter 6 gives the computational results and presents the execution times of the algorithms and the objective quality of their segmentation results.

Finally, Chapter 7 concludes the thesis and states some possible future work.

CHAPTER 2

PROBLEM DEFINITION

2.1. INTRODUCTION

A fundamental problem in image analysis and pattern recognition is image segmentation. The problem is partitioning an image into semantically meaningful connected regions which have similar properties in some sense e.g. similar color or gray level, similar texture; and which are different from neighboring regions [5][9].

Algorithms that have been proposed to solve this problem can be classified into two groups. First, one is to define the regions by boundary detection based approaches. These techniques partition the image by finding out closed boundary contours. Second group of algorithms consist of region growing and clustering based approaches. In these approaches regions are defined by the union of their component pixels. Region based approaches can be further

classified into two groups, one using global and the other using local information [5, 9].

2.2. GRAPH THEORY AND IMAGE ANALYSIS

Graph theory is the study of graphs and their applications. A graph ***G(V,E)*** is a set consisting of set of vertices, V, and set of edges, E. The number of vertices is called the order of graph and denoted by $|V|$. The number of edges is called the size of graph and denoted by $|E|$. Vertices V_i and V_j are said to be adjacent to each other by *links* $E_{i,j}$, for $i \neq j$, and where V_i and V_j are the vertices that the link connects. A weighted graph has an associated weight for every edge $E_{i,j}$, and for every vertex V_i, in the graph. The complete graph is a simple graph in which every vertex is adjacent to every other. A *subgraph* of a graph G is a graph whose vertex and edge sets are subsets of those of G. A *path* is a set of successive vertices in which each vertex is connected to the next by an edge in the graph. A cycle in a graph consists of a sequence of successively incident edges and their end vertices, where the terminating vertices are identical. A *tree* is a connected set of paths having no cycles. A *forest* is a set of *trees*. A *spanning tree* is a tree, which is a subgraph containing all vertices and having no cycles. The *shortest spanning tree* of a weighted graph is a spanning tree such that the sum of its link weights is minimum for all possible spanning trees. Extensive information about graph theory can be found on various resources such as [18, 6].

2.2.1. MAPPING IMAGES ONTO GRAPHS

To analyze images by graph theory, the original image must be mapped onto a graph. It can be done simply by mapping each image pixel onto a vertex of the graph. Pixel intensity value (for gray level images) can be assigned to the vertex weight. A vertex in this graph can be linked to any other vertex but a meaningful connection is to link each vertex to its nearest neighbors. Either four

or eight neighbors can be considered. Figure 2.1 shows a mapping of a 5x5 image onto a four neighbor connected graph.

Weights associated with each link are based on some property of the pixels that it connects, such as their image intensities [10]. If the link weights are defined as the absolute value of the difference between the weights of the vertices that they join, then the link weight is a measure of similarity between the two vertices, and hence between the two corresponding pixel intensities [5].

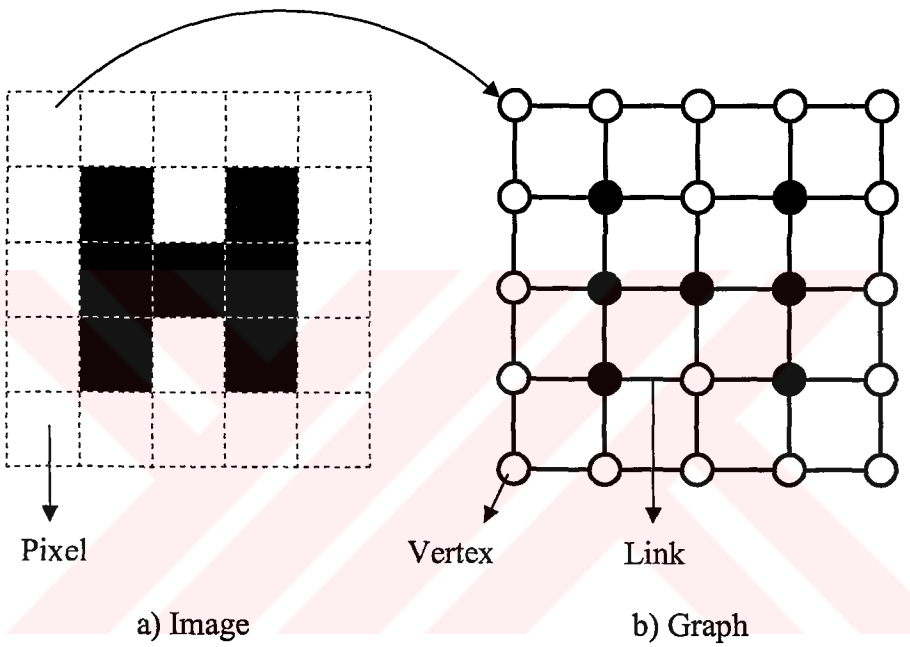


Figure 2.1. Mapping an image onto a four connected graph

2.2.2. CONSTRUCTING MINIMUM SPANNING TREES

Constructing the minimum spanning tree of the graph obtained from the image is used commonly in graph based image segmentation. Minimum spanning tree (MST) of a graph is a sub-graph that contains all the vertices and the sum of its link costs is minimum for all possible spanning trees. If a forest forms part of a MST then the complete MST can be found by successively adding new links to it.

The MST therefore tends to include links with low weights. More costly links are only included when there is no alternative. The MST of an image is shown in Figure2.2.

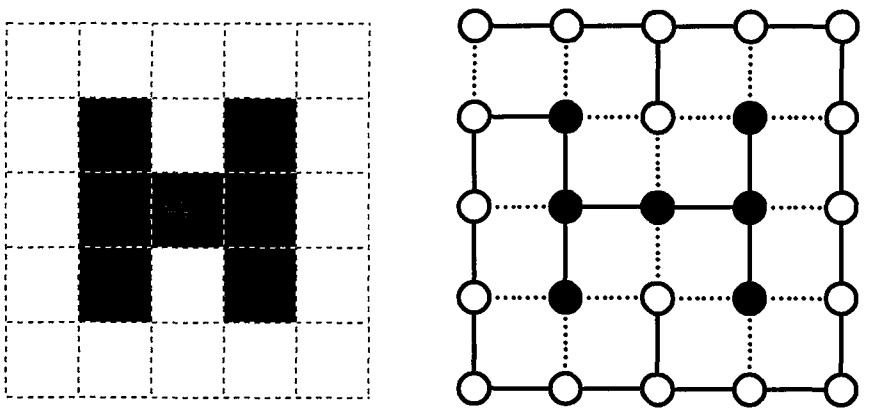


Figure2.2. Constructing minimum spanning tree from the image. Bold links are included in the MST, dashed ones are not included.

2.2.3. FORMING PARTITIONS FROM SPANNING TREES

To form partitions from the spanning tree representation of an image, some links included in the spanning tree must be cut. By cutting its links, a spanning forest is obtained and partitions of the image are formed. To form 'n' number of partitions 'n-1' cuts must be performed. This is because of the fact that each cut produces a subtree which is disjoint from all of the other previously formed subtrees. Because of this property every subtree represents a region of the image. The number of cuts depends on the user. Once the spanning tree representation is obtained, region number can be selected as much as desired up to the pixel number of the image. User determined region number is not

applicable to many other segmentation methods such as threshold based ones. Figure 2.3 illustrates obtaining two regions by cutting a single link of the spanning tree of the image.

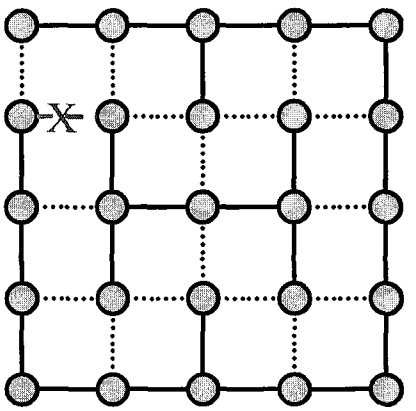


Figure 2.3. Cutting a single link of a spanning tree to obtain two regions.

CHAPTER 3

GRAPH THEORETIC APPROACHES FOR IMAGE ANALYSIS

Table 3.1 shows graph theoretic algorithms having applications to image analysis and states their advantages and disadvantages.

Table 3.1 Application of graph theoretic algorithms to image analysis [13]

Graph-Theoretic Algorithms	Applications to Image Analysis	Pros and Cons
Breadth-First Search and Depth-First Search	• Simple distance functions • Connectivity • Euler number computation for shape description	• Fast and simple • Implement by queue or stack respectively • Distance function very restricted

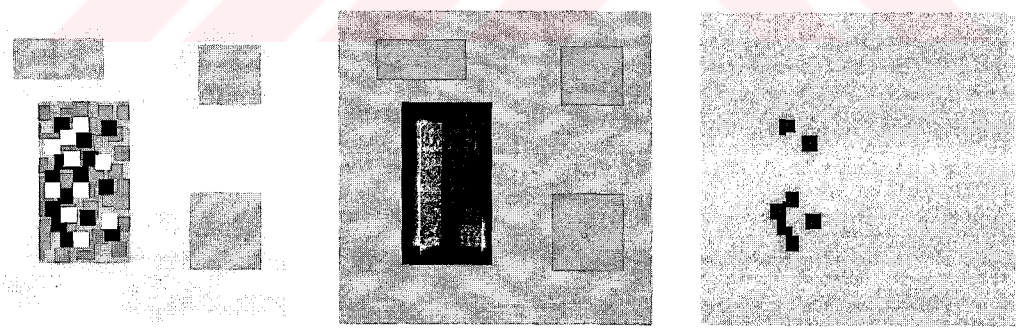
Table 3.1. Cont'd

Dijkstra, Fast Marching (hybrid graph/PDE), Dynamic Programming (generic optimization tool)	• Euclidean distances • Weighted (geodesic) distances • Segmentation (2D images only) • Tracking • Stereo matching with 1D constraints • 1 parameter (1D) sequence optimization	• Relatively simple • Very flexible • Limited to one parameter (1D) sequences; can't be extended to optimal surfaces
Minimum cuts	• Segmentation (any image dimension) • Feature clustering • Minimal surfaces • Stereo matching with 2D constraints	• Fairly complicated • Algorithms are relatively slow • Lots of research going into this area • Extensible to N-1 degree manifolds in N dimensions
Optimal Spanning Trees/Forests	• Watersheds • Seeded region growing (variant) • Feature clustering • Segmentation	• Greedy algorithm (Priority First Search) • Simple, but can 'leak' through one bad edge in the graph (not very robust to noise or errors)

Table 3.1 Cont'd

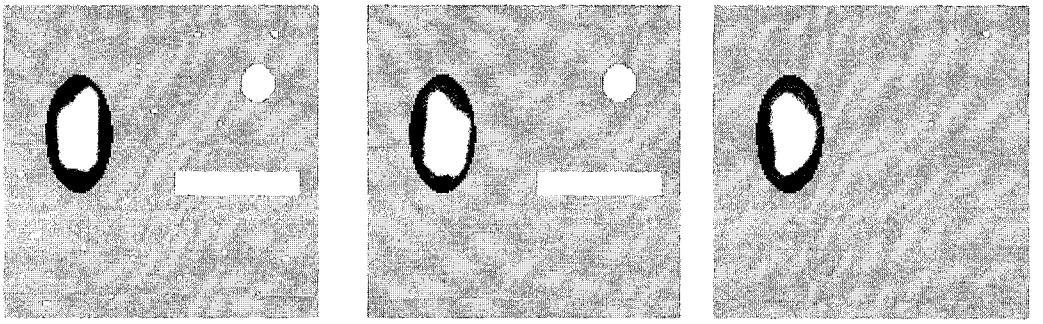
Graph isomorphism/homomorphism	• Character recognition • Recognition of shapes by topology only	• No polynomial algorithms exist yet (undecided problem)
Optimal Matching and Assignment	4. Multiple object tracking	• Simplified minimum cut problem

Following the above general classification, a literature on image segmentation based on graph methods is presented in this chapter. Zahn's [11] segmentation method depends on cutting the minimum spanning tree of an image from edges having largest weights. This algorithm is highly noise sensitive and inadequate to determine high variability regions and usually results in incorrect partitioning. This is illustrated by the example in Figure3.1 and Figure3.2.



a) b) c)

Figure 3.1. a) Original image will be partitioned into five regions, b) Expected segmentation of the image, c) Image is segmented into five regions by cutting most costly four links after constructing the minimum spanning tree of the image graph. High variability region is partitioned into several regions.



a) b) c)

Figure3.2. a) Original image will be partitioned into four regions, b) Expected segmentation of the image, c) Image is segmented into four regions by cutting most costly three links after constructing the minimum spanning tree of the image graph. Noisy pixels are interpreted as regions and objects.

In order to solve this problem Urquhart [12] proposes to normalize the weight of an edge using the smallest link weight of the links incident on the vertices touching that edge. However, the application of this method to image segmentation showed that the smallest weight edge alone is not adequate for a reasonable segmentation. It can be realized by looking at the high variability region pixels in Figure3.1. Many pixels in the region have some highly similar neighbors [10].

Morris *et al.* [4, 5] stated that these methods do not depend on global information and do not achieve good segmentation. Hence, he proposed a method called Recursive Shortest Spanning Tree (RSST) Algorithm and claims that this approach improved the segmentation results considerably. RSST has a hierarchical structure and due to this property, any number of region representations of an image can be chosen. Zeng [14] emphasizes that the generated subtrees obtained by RSST do not properly represent the regions in the order of perceptual significance. He introduced homogeneity and saliency information in his modified RSST algorithm.

RSST is used in many applications. In [6] these applications are summarized pretty well. Researchers used RSST algorithm in segmented image coding [15] and segmented video coding [16]. Morris and Constantinides in [22] propose a progressive image coding scheme using RSST. Kwok *et al.* extended RSST to temporal segmentation [23] and temporal decimation [24] for video processing applications. Alatan *et al.* [25] applied RSST in interactive multimedia services to motion vector-based segmentation. RSST is also used for object extraction [26] and for object segmentation and tracking [27, 28] in video applications. Tuncel and Onural [29] applied RSST to 2-D affine motion modeling and video content representation using its multiresolution implementation by [30] and [31]. RSST has another application area used for multiresolution segmentation and is called M-RSST. This algorithm is applied for content-based face detection [32], an active contour-based video object segmentation scheme for stereoscopic video sequences [33], an efficient unsupervised content-based segmentation in stereoscopic video sequences [34], and video databases [35]. Vlachos and Constantinides proposed RSST for color images [36], in which color components of red, green, and blue are translated to the cost function. Before this, RSST algorithms and its applications were based on grayscale images [6].

Efficient implementations of RSST were reported by Kwok and Constantinides [6, 7, 8]. In [8], they proposed a tailored data partitioning strategy to assign jobs to processing elements in their parallel algorithm; they claim that their parallel implementation is cost-optimal, however they did not implement it since their proposed algorithm's architecture is a Concurrent Read Exclusive Write Parallel Random Access Machine (CREW PRAM) which is not available today. They simulated their proposed algorithm for one processor, which implies a sequential run. In the fast RSST implementation, they claim that they speed up the RSST algorithm from the complexity of $O(n^2)$ to $O(n)$ in the worst case, where n is the number of vertices in the graph [7]. In our study, fast implementation is achieved by removing the sorting algorithm from the

conventional algorithm. However, our study showed that accelerating only sorting part does not improve the complexity of RSST that much. Outputs of these efficient RSST algorithms [7, 8], are truncated versions of RSST, which is different from the RSST generated by [4], [5], and [6]. It is worth noting that the literature survey shows that the studies on RSST algorithm use only absolute intensity difference as the link weight cost. No other cost function is mentioned and we are currently unaware of a study, which examines the link weight cost function effects on the performance of the algorithm.

CHAPTER 4

RECURSIVE SHORTEST SPANNING TREE ALGORITHM

This chapter presents the Recursive Shortest Spanning Tree (RSST) Algorithm, its fast version and a simple modification on RSST. A second link weight cost function, which is not used in the previous algorithms, is applied to the conventional RSST and its performance analysis is presented. Lastly, a distributed approach is proposed and analyzed afterwards.

4.1. SEQUENTIAL ALGORITHMS

4.1.1. CONVENTIONAL RSST

The Recursive Shortest Spanning Tree (RSST) Algorithm proposed by Morris *et al.* [4,5] was used for image segmentation and edge detection. The flowchart of RSST is given in Figure4.1. The first stage of the RSST algorithm is mapping the image onto a graph as described in section 2.2. Initially, each pixel

corresponds to an individual region and each region is represented as a node (vertex) on the graph. Pixel intensity values are assigned to vertex weights. Link weights are assigned by the cost function, which can be a function of the vertex weights and the sizes of the connected regions [6]. Morris *et al.* [4,5] define the link weights as the absolute value of the difference between the weights of the vertices that they connect. Therefore, link weight becomes a measure of similarity between two regions.

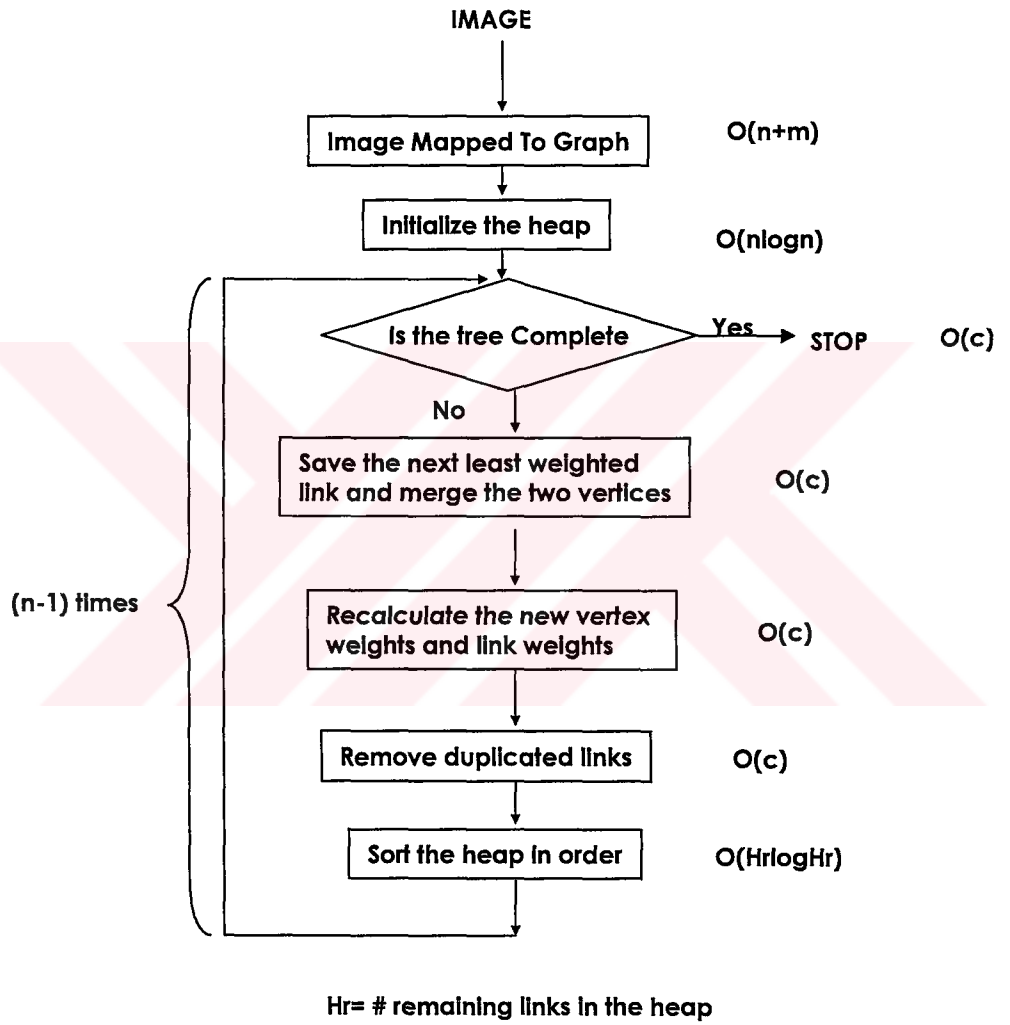


Figure 4.1. Flowchart of conventional RSST Algorithm

After mapping the image onto a graph, next comes the process of finding similar vertices. This is done by constructing the minimum spanning tree from the given graph with least weighted links. Firstly, all the links are sorted

according to their link weights. Since one of the most efficient sorting algorithms is the heap sort, links are stored in a heap structure. The link at the top of the heap, having the minimum cost is then chosen. The chosen link is saved as it is the part of the SST of the graph and two connecting regions are merged.

While evaluating the new link weights and vertex weights, sizes of regions and their weights are considered. For example after merging the region R1 with N1 pixels with a vertex weight equal to W1 with a region R2 with N2 pixels and W2 weight, the vertex weight of the resulting region R' is defined as follows:

$$W' = [\,(W1 \times N1) + (W2 \times N2)\,] / (N1 + N2) \quad \text{.....(4.1)}$$

$$N' = N1 + N2 \quad \text{.....(4.2)}$$

Links connecting the newly merged region to other regions must be updated since their costs are changed after merging as shown in Figure 4.2.

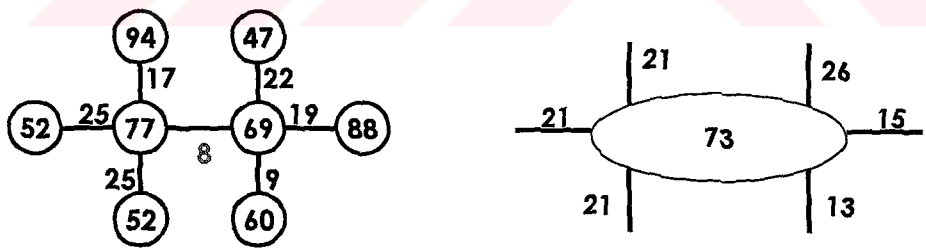


Figure 4.2 Merging process. a) Before merging, two regions, vertices, with weights 77 and 69 will be merged, b) After merging vertex weights and link costs are updated, two pixels represent one region.

In the merging process, duplicated link removal must be considered to avoid cycles in the MST. When two regions are connected with more than one link, redundant ones are called duplicated links. This is shown in Figure 4.3.

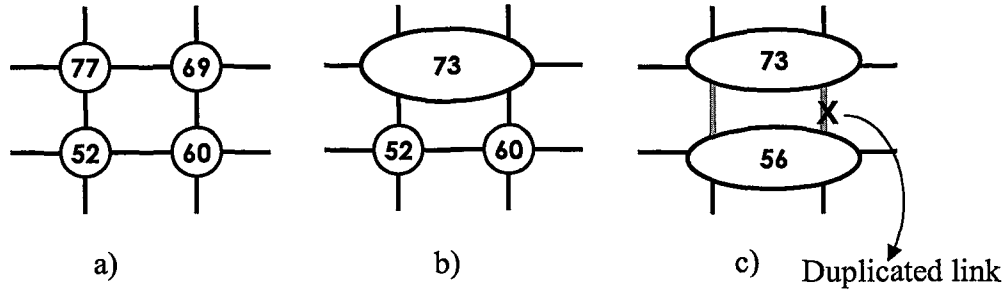


Figure 4.3 Duplicated link formation and removal. a) Before merging, b) After one merging process, c) Second merging forms a duplicated link, which must be removed.

After merging and duplicated link removal process, links must be sorted again. Due to the removal of duplicated links and saved links, rearranging of the heap is inevitable. This is the most costly and time consuming, part of the algorithm.

The merging, duplicated link removal and heap rearrangement processes are repeated until the spanning tree representation of the image is completed. Hierarchical representation of the image is obtained by noting down the order of the saved links. Now by cutting this spanning tree representation of the image from N links, N+1 regions are obtained. The user can define the number of regions to be segmented.

4.1.2. FAST RSST (FRSST)

The conventional RSST is computationally inefficient but gives good segmentation results. Therefore, some research has aimed at obtaining fast implementations of the RSST. Researchers have claimed that the execution time

of the RSST is heavily dependent upon the sorting algorithm. Therefore, to speed up the algorithm sorting part is generally modified. Fast RSST algorithm [7] depends on this idea.

The flowchart of the Fast RSST algorithm is given in Figure 4.4. The algorithm is as follows: Image is mapped to the graph in the same way as in RSST but only link weights are truncated to integer values. Instead of using a heap structure stacks are used. All links are stored in the stacks, which are called link weight stacks (LWS). In *LWS(i)*, only links with weights equal to *i* are placed. So the maximum possible number of stacks is equal to the upper range of the link weight function. If link weight function is the absolute difference of region intensities then the biggest possible link weight stack is *LWS(255)* (for gray level images). Fast RSST algorithm uses links in an ascending order starting from *LWS(0)* to the largest possible *LWS(i)*. Working stack is defined as the stack with the least link weight. [8].

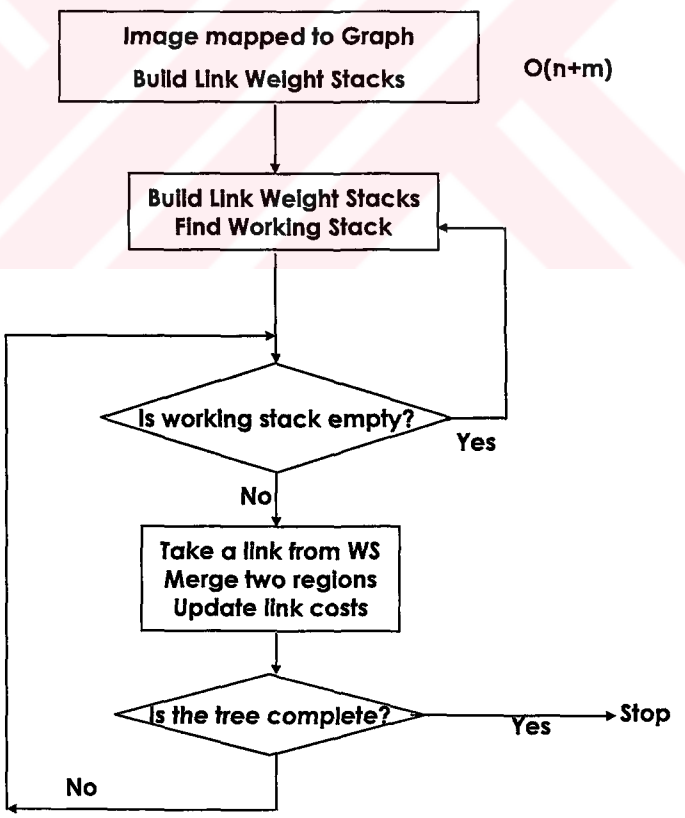


Figure 4.4 Flowchart of the Fast RSST Algorithm

After building the link weight stacks, merging process starts. Finding the least weighted link in this case is easy and can be done in constant time. It is found in the working stack. Any of the links in the working stack can be chosen and inserted in the merging process. Merging is done as in RSST. The process is illustrated in Figure 4.5.

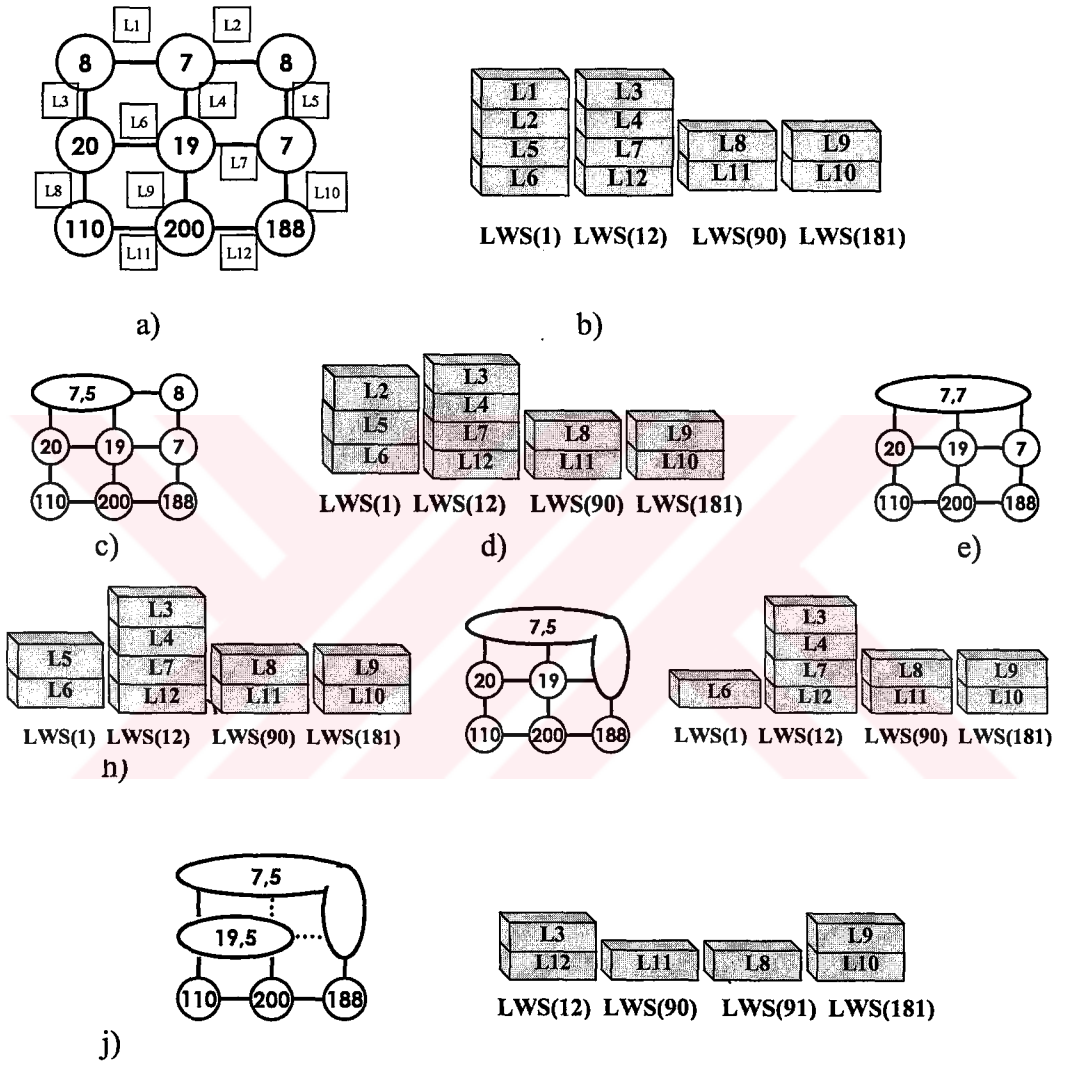


Figure 4.5. a) A 3x3 image, b)Associated link weight stacks, Working Stack: *LWS(1)*, c) One link is chosen from WS and connecting regions are merged, d) Link weight stack states, e) Resulting graph after second merging, f) Link weight stacks state after second merging, g) Resulting graph after third merging, duplicated links are formed, h) Link weight stack states after third merging, i) Resulting graph after fourth merging and WS is now empty, j) Link weight stack states after fourth merging and working stack becomes *LWS(12)*.

Updating link weights are postponed until the working stack becomes empty. When working stack becomes empty, all affected link costs are updated and placed into appropriate stacks. This is shown in Figure4.5. While rebuilding the stacks, working stack determination is done. Again this procedure is repeated until the spanning tree representation of the image is completed.

In the fast implementation of the algorithm, sorting part is done more efficiently than the conventional RSST algorithm but output quality degrades slightly. To improve the output quality while keeping this stack structure we propose a small change in the fast RSST algorithm.

4.1.2.1. Modified Fast RSST (M-FRSST)

The flowchart of the modified fast RSST algorithm is given in Figure 4.6.

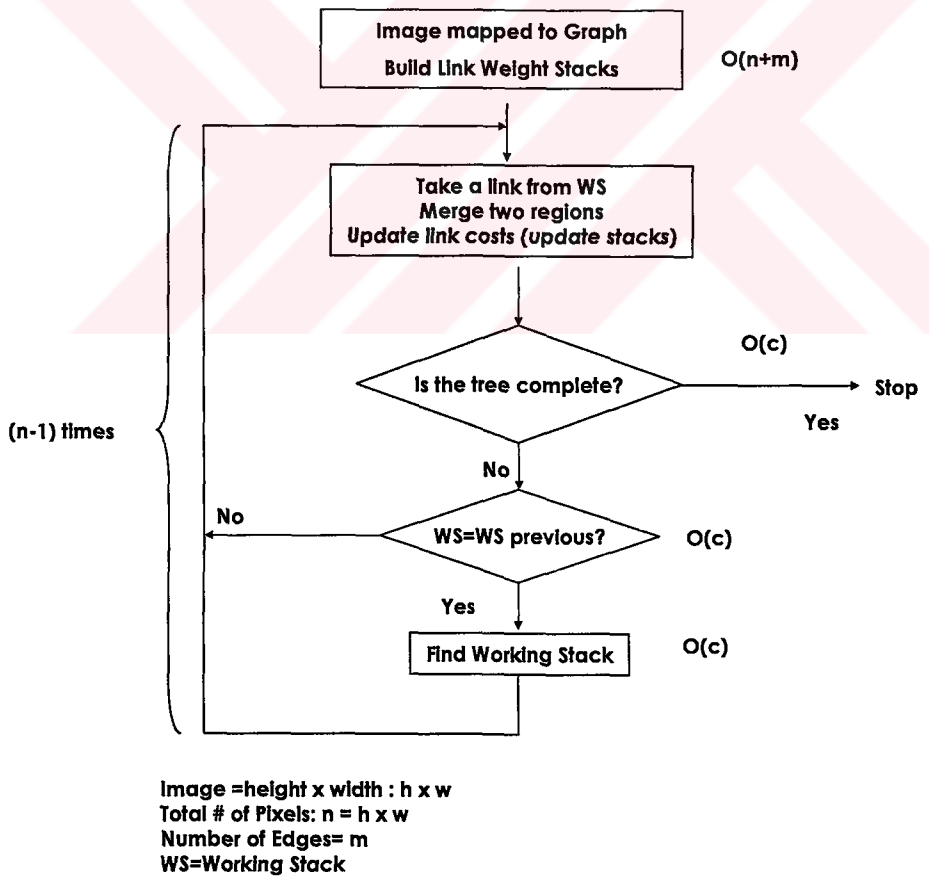


Figure 4.6. Flowchart of the modified Fast RSST Algorithm

This approach is very similar to the first one except the process of rebuilding the link weight stacks. In the first approach, stacks are required to be rebuilt after the working stack becomes empty. In the second approach, updating vertex weight and removing duplicated links processes are same as in RSST. That is after every merging process link weights are needed to be rebuilt. Additional control on the working stack determination is included. In example in Figure 4.7 before merging working stack is *LWS(1)*, after merging working stack becomes *LWS(0)*.

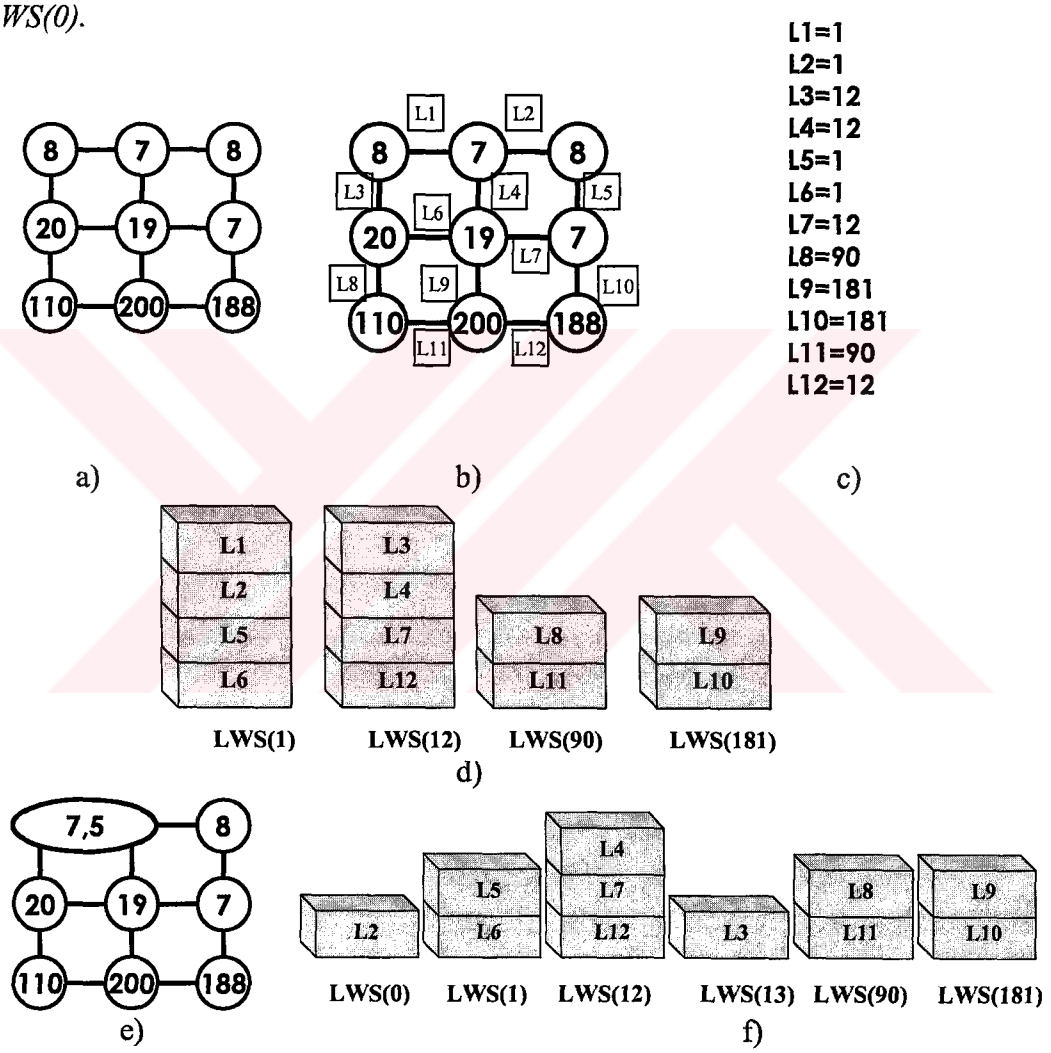


Figure 4.7. a) A 3x3 image, b) Links are numbered, c) Link costs are evaluated, d) Links are stored in *LWS(i)*'s, e) Link1 is chosen and two regions are merged, f) States of stacks after merging.

4.1.3. MODIFIED RSST (MRSST)

To improve the run-time performance of the RSST algorithm, we propose a simple change in the algorithm. In the conventional algorithm heap is rearranged after each merging process. Rearrangement is done by finding and updating the place of the modified links in the binary heap. The new place of each link in the heap, which is a neighbor of the recently merged region, is found in this process. What we propose is to bring a condition for the relocation of a link in the heap structure. If the link weight does not change too much with the merging process, relocation is not necessary. If the link weight is changed reasonably, than a rearrangement of the heap structure is needed. The minimum change in the link weight to relocate the link in the heap is determined by experimentation in the computational study.

4.1.4. OPTIMUM COST RSST (ORSST)

With segmentation, pixel values in each region changes according to Equation 4.1. The measure of success in segmentation is usually an error measure between the original image gray value matrix $G[x,y]$ and the segmented image gray value matrix $\hat{G}[x,y]$. It is called Mean Square Error and given by Equation 4.3. No mathematical error function is known that corresponds to human perceptual assessment of error [21].

$$\sigma^2_{error} = \frac{1}{m\,x\,n}\sum_{i=1}^{n}\sum_{j=1}^{m}(g_{ij}-\hat{g}_{ij})^2 \quad \text{.....(4.3)}$$

To minimize this function at every merging step, costs of links neighbouring to the regions should be updated and link cost *d(reg1,reg2)* must be defined as follows:

$$d(reg1, reg2) = \sqrt{\frac{n1.n2}{n1+n2}}\,|m1 - m2| \quad \text{.....(4.4)}$$

where n1 and n2 corresponds to region sizes in terms of pixel number, m1 and m2 corresponds to mean intensity values of the regions.

Detailed information about this MSE minimization can be found in [3]. If the link cost is chosen as defined by equation 4.4. RSST minimizes mean square error between the original image and the segmented one. This may make the RSST more advantageous among all other segmentation algorithms. Up to this point, the previously mentioned algorithms seen in literature use absolute intensity difference as the link weight cost function. To our best knowledge, no RSST report about the effects of using the above cost function exists. Therefore, we performed experiments using this cost and its run-time performance figures and segmentation quality results are presented in chapter six.

4.2. DISTRIBUTED RSST

A parallel RSST algorithm has already been proposed by Kwok and Constantinides [8]. It is claimed that this parallel implementation proposal is cost optimal such that the complexity of the RSST algorithm is reduced from $O(n^2)$ to $O(n)$ in the worst case where n is the total number of vertices in the image graph [8]. A CREW PRAM (concurrent-read-exclusive-write parallel random access machine) model is assumed for their algorithm [8], which is not a realizable computer model for today. Hence, Kwok and Constantinides propose the theory of their algorithm and present results only for the case where the number of processors is equal to one. This means that the algorithm runs sequentially. Due to the unavailability of the PRAM architecture, current thesis work proposes and implements a practically possible distributed algorithm instead.

In the literature, we are faced with the idea that if the image is physically partitioned and segmented in parallel, then the resulting segmentation will not be optimal and a suboptimal solution will be obtained. This judgment is also presented in [8]. However, to the best of our knowledge, there exists no report, which studies the extent of this sub-optimality. It may be worth investigating whether a simple image partitioning and solving the partitions with ORSST and then combining the separate processor results, generates an acceptable level of degradation in segmentation quality or not. If there exists a speed up against a slight quality decrease, this approach may still be acceptable in various applications where there is a need for segmentation of large images.

In distributed computing, network traffic affects the speed of the algorithm greatly. Communication types (blocking or non-blocking), message size, bandwidth of the network are among the parameters that affect the complexity and performance of the algorithm. The communication between the processors should be kept at minimum to minimize the degradation in the timing performance.

In our distributed algorithm there is a master and several slave processors. The algorithm divides the image into equal size partitions. The number of partitions is equal to the number of available slave processors. Each partition is processed by a single slave. Each slave uses link weight cost function as defined by Equation 4.4. Our distributed algorithm is briefly stated as follows:

```
1. (Master) Divide the image into equal size regions in
   such a way that the partitions have overlapping parts
   with neighborhood partitions.
        Image height: h
        Image width: w
        Number of processor: p
```

Overlapping area height: v

Size of a region: (h/p + v).w

2. (Master) Assign each partition to a different processor.
3. (Slave) Each processor runs the RSST algorithm on the given image partition to obtain the shortest spanning tree representation of the corresponding image partition (i.e. number of segmentation region is equal to one).
4. (Slave) Each processor sends its own links that are included in the spanning tree of that partition to the master processor with associated link weights that corresponding to those set at the time of their selection to the tree.
5. (Master) Master processor examines the received links, constructs a new but reduced graph from the received links only and selects the least weighted link among the received links. The chosen link is saved as part of the final SST of the whole image graph and the two connecting regions are merged in the reduced graph. Then the duplicated link removal process is performed. Next link is searched among the remaining received links set and this selection and merging process is repeated until the desired number of region representation is obtained.
6. (Master) Master maps the graph onto the image.

Figure 4.8 shows image partitioning.

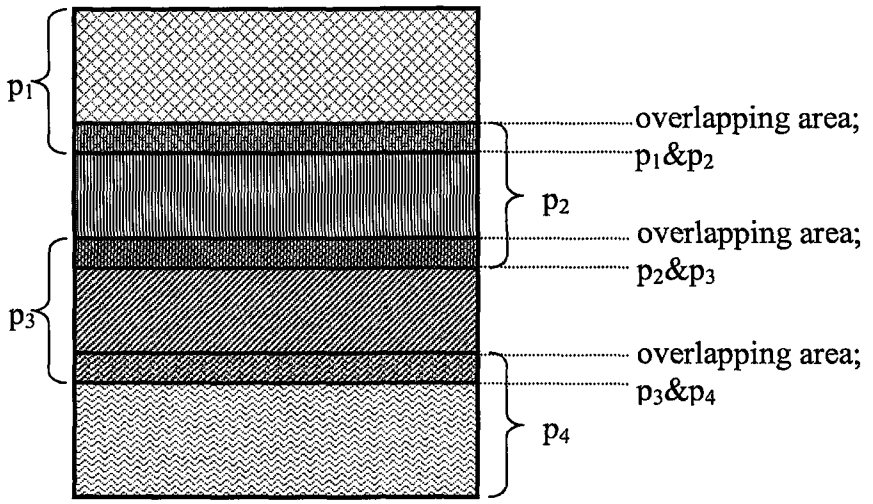


Figure 4.8 Division of the image into equal size partitions

Slave processors have smaller size images so they compute their spanning trees in a shorter time. Master processor initially constructs the whole data structure from the whole image. After receiving the spanning trees of the slave processors, it reduces this data structure. This is equivalent to constructing a new graph consisting of received links only. Links that are not included in any of the spanning trees of the slave processors can be removed from the original graph. This minimization decreases the duplicated link removal complexity in the final reconstruction phase (step 5) considerably. When this algorithm is compared with the original RSST, the heap sort operations are expected to be reduced. The division of the image into partitions by the distributed RSST results in the loss of global information, i.e., each processor has the knowledge of its own partition only. This drawback results in a different spanning tree representation of the image, which might result in a different segmentation. In chapter 6, besides the timing issues, the segmentation quality of the distributed RSST algorithm is also evaluated by checking the SNR figures.

Figure 4.9. illustrates the algorithmic steps for the one master, three slave case. The image, which will be processed, is in the shared memory. Firstly, the master processor informs the slaves about their starting and ending pixels (Figure4.9.i). This information transfer is not done in the form of message passing but a calculation of pixel ids using the id of slave processor is carried out. Pixel assignment is done considering the overlapping condition. We use two overlapping rows in our executions. Increasing the overlapping area will decrease the speed up because of increasing slave time. All slaves start their processing at the same time. With this starting and ending pixel information, slave processors individually construct their conventional RSST data structures (Figure 4.9.ii). This procedure is completely the same as conventional RSST. The number of regions are used as "one" for all slaves. This makes them to build the whole spanning tree of their own image parts. Choosing the least weighted link and the merging processes are very similar to the conventional algorithm. However during the merging process the slaves store the link's id and the accompanying link cost one after the other to an array, called the spanning tree information array (Figure 4.9.viii). During this time, the master processor waits in the idle state. It waits for the spanning tree information arrays from the slaves. After constructing their own spanning trees, the slaves send their spanning tree information arrays to the master processor (Figure 4.9.iv).

Receiving all spanning tree information arrays from the slave processors, actual work for the master begins. It already has the information of all the graph links. Because of the reconstruction approach proposed, which is to find a minimum spanning tree using the received costs on a graph consisting of the received spanning tree links only, most of the links in the original graph become redundant. These redundant links are then deleted from the master's data structure (Figure 4.9.vi) in the graph reduction procedure.

Up to now there is no region information obtained by the master but it only has the recursive shortest spanning trees of the image partitions that are

constructed independently. Link selection and region merging is the next step at this point. Links and accompanying costs of the spanning tree information arrays sent by slaves are in sorted order and therefore an efficient linear search among the arrays is possible. The minimum cost link is then selected easily. Merging and duplicated link removal procedure is done next as is done in the conventional algorithm. Above link selection, merging and duplicated link removal procedure is repeated until we reach the desired region number. Then the graph can be mapped onto the image by assigning every pixel in a region the average gray values of the pixels forming that region.

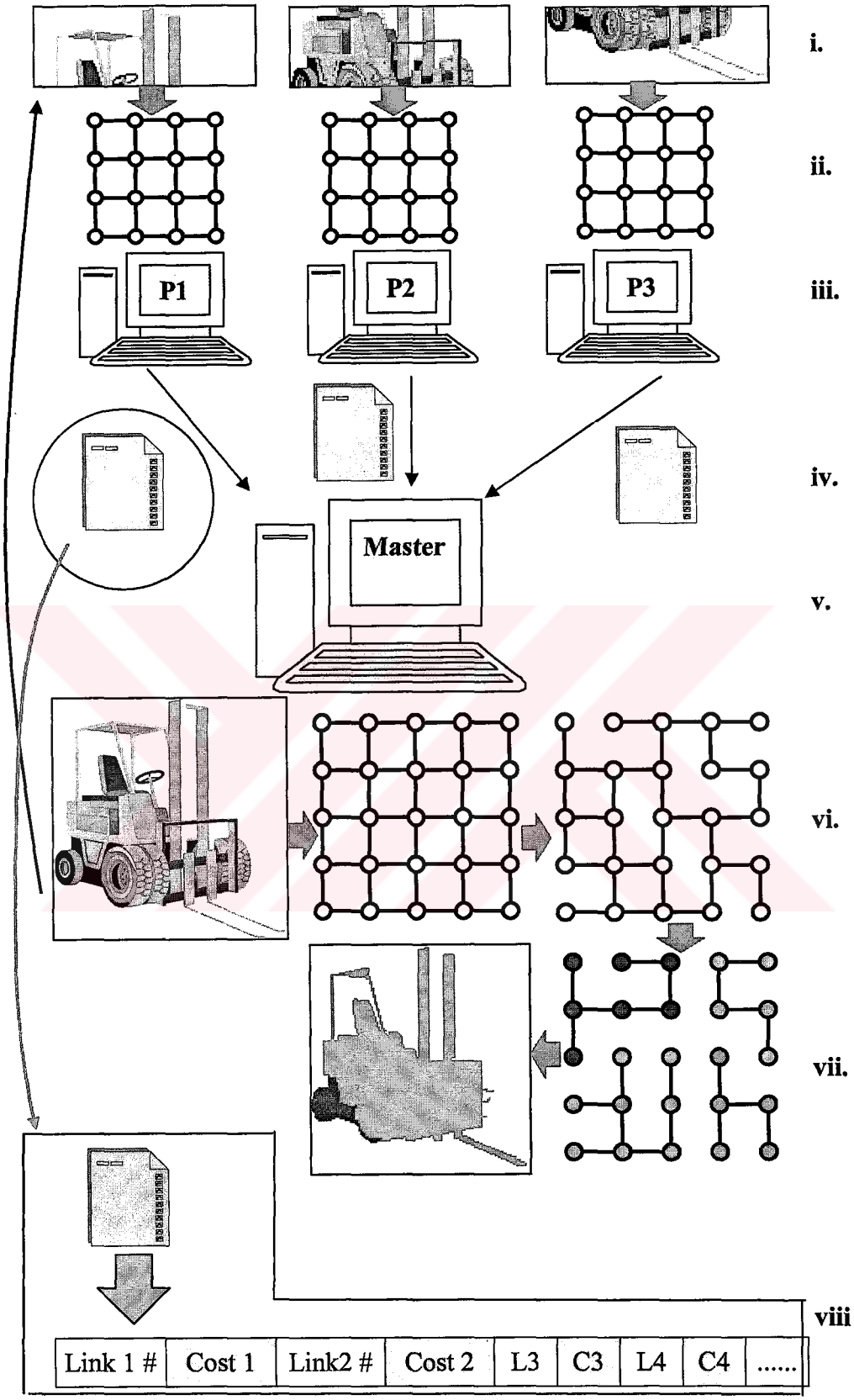


Figure 4.9. Distributed Algorithm's steps for the three slave processors case

CHAPTER 5

IMPLEMENTATION

This chapter presents the implementation details of the conventional RSST, Fast RSST, Modified RSST, Optimum Cost RSST and Distributed RSST algorithms. Microsoft Visual C++ 6.0 is used as the programming language and software development environment. OpenCV libraries are included into the projects for image reading and writing operations. OpenCV is Intel® Open Source Computer Vision Library, which is a collection of C functions and a few C++ classes that implement some popular algorithms of Image Processing and Computer Vision [17].

In this study implementation of the Recursive Shortest Spanning Tree for COST 211 AM is used. It was implemented in 1997 by Patrick Mulroy from BT Labs, Paulo Villegas from Telefonica I+D and with help from Ertem Tuncel, Bilkent University. Data structures, functions and algorithm used in this project are given in Appendix A, B and C.

In Fast RSST implementations, the basic data structures are inherited from the above RSST implementation but stacks are used instead of the heap structure to speed the algorithm up.

Modified RSST has completely the same data structure with the original one. Only a simple condition is added to the heap sorting part.

In some distributed memory systems, each processor has its own local memory and this memory can be accessed only by itself. Data transfer from one processor to another is performed over a network by message passing. (Figure 5.1.)

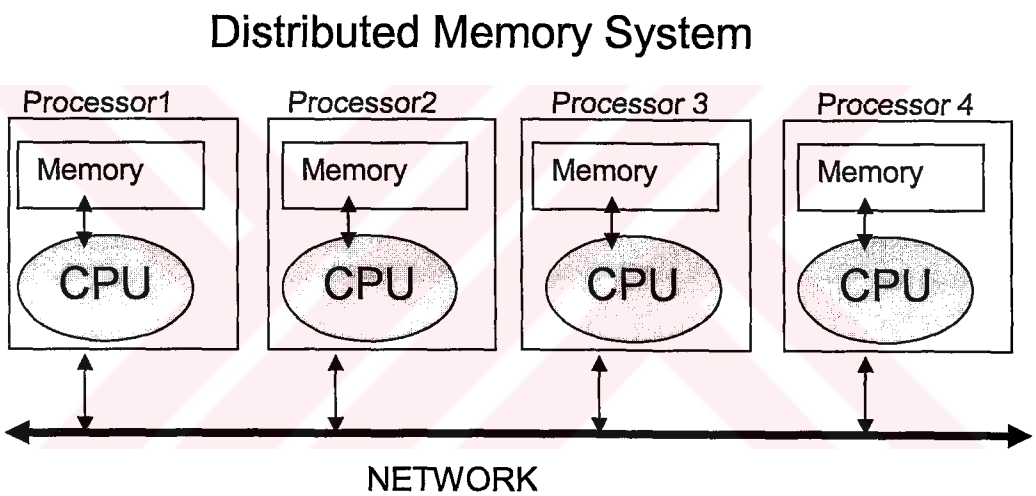


Figure 5.1. Distributed Memory System

The distributed RSST developed in this work uses Message Passing Interface (MPI) standard for processor communications. In this study ANL/MSU (Argonne National Laboratory /Mississippi State University) MPICH, which is a freely available, portable implementation of MPI is used [19]. MPI is a specification for message passing libraries, which aims to standardize these libraries. In MPI programs number of processes are static, no new processes can be included dynamically during the run. It can be used with C and FORTRAN. It has over 115 routines to accomplish send, receive and other message passing

functions. MPI provides both blocking and non-blocking communication types. In blocking communication for send operations data must be safely placed to the receiver side before the system buffer can be reused. Similarly for receive operations data must be copied to the system buffer safely before it can be used. This is a simple acknowledgement based system. Both of the receiver and sender cares about the message whether it is sent and received correctly or not. Distributed RSST implementation should use blocking send and receive operations since the system can not tolerate data losses.

MPI firstly uses MPI_Init subroutine; which is called by every processor; to guarantee the processors in the environment are ready to run the MPI. Every processor has an integer identification number supplied by the system. This unique number is owned when the process is initialized and can be used to specify the source and destination of a message. An MPI program should call a finalization subroutine when all the communications are completed. MPI_Finalize routine cleans all MPI data structures after which no other MPI calls can be made. A simple "Hello World" program is depicted in Figure 5.2.

```
#include "mpi.h"
#include <stdio.h>
int main( int argc, char *argv[] )
{
MPI_Init( &argc, &argv );
printf( "Hello, world!\n" );
MPI_Finalize();
return 0;
}
```

Figure 5.2. A simple MPI Program written in C language

If the code in Figure 5.2 is executed all processors in the environment print the sentence "Hello World" onto the screen in a random order. This is an example of a single program on a single data. Multiple instructions multiple data (MIMD) programs can be implemented using MPI. An example of a MIMD program is shown in Figure 5.3.

```
#include "mpi.h"
#include <math.h>
#include <stdio.h>
void main(int argc, char *argv[])
{
      int i, spd, n, myid, numprocs;
      MPI_Init(&argc,&argv);
      MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
      MPI_Comm_rank(MPI_COMM_WORLD,&myid); // each processor is
                                          //assigned with identifier
      if(myid==0)  // specific processor is selected
            {
            spd=0;
            for(i=0;i<5;i++)
                  spd=spd+i;
            printf("My id is %d and sum is %d\n",myid,spd);
            }
      if(myid==1)  // specific processor is selected
            {
            spd=1;
            for(i=2;i<=5;i++)
                  spd=spd*i;
            printf("My id is %d and product is %d\n",myid,spd);
            }
      if(myid==2)  // specific processor is selected
            {
            spd=2;
            for(i=0;i<5;i++)
                  d=spd*spd;
            printf("My id is %d and exp is %d\n", myid, spd);
            }
      MPI_Finalize();
}
```

Figure 5.3. A MIMD MPI program example

Possible output of the program will be as follow:

```
My id is 2 and exp is 64
My id is 0 and sum is 10
My id is 1 and product is 120
```

Our implementation is also a MIMD program since the master and the slave processor do different tasks. Selection of master and slave processors is done same as is done in this example, i.e. by stating its rank. This strategy of programming allows single code construction for all processes.

Since MPI is the available distributed implementation facility in this work, communication between the master and the slave processors are only

blocking send-receive operations. Therefore, a slave waits the master if at the same time the master is communicating with another slave. Similarly, the master has to wait for a slave processor if the slave has not finished its work yet. Thus selecting similar performance slave processors will utilize the processors more efficiently, decreasing idle processor time. The master waits for the spanning tree information arrays from the slaves in the order of their processor ids. Therefore, the worst case regarding the speed of the overall distributed algorithm is the case where the first processor is the slowest among others.

CHAPTER 6

COMPUTATIONAL RESULTS

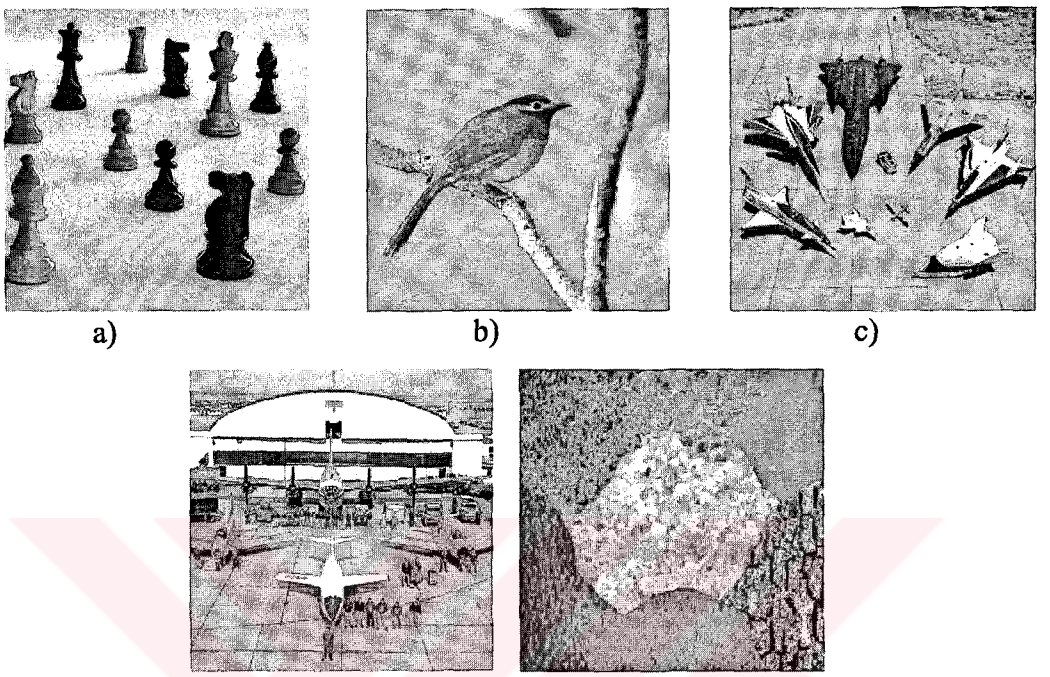
This chapter presents the comparison of the performances of the RSST, Fast RSST, Modified Fast RSST, Optimum Cost RSST and Distributed RSST algorithms. In order to compare their performances, two criteria, namely execution time of the algorithms and objective quality of the segmented images (Signal to Noise Ratio (SNR)), are considered.

6.1. COMPARISON OF SEQUENTIAL ALGORITHMS

6.1.1. EXECUTION TIME COMPARISON

The aim of these experiments is to compare the run-time performance of the algorithms with respect to increasing image size. In these experiments five different images are used. Each image is resampled at 20 different sizes and has a maximum size of 1,440,000 pixels (1200x1200). Images used throughout the thesis are shown in Figure6.1 a,b,c,d, and e. The number of regions to be

segmented is selected to be one. With this selection, a spanning tree of the whole image is constructed at the end of each run. Execution time of the corresponding image size is given as the average of the execution times of these five images.



a) b) c)

d) e)

Figure 6.1. Images used throughout the thesis, a) Image No:1, b) Image No:2, c) Image No:3, d) Image No:4, e) Image No: 5

Algorithms are run on Intel Pentium 4, 1.8GHz processor and 512 Mbytes RAM. The run-time performances of the algorithms are plotted in Figure6.2 and Figure6.3. Figure 6.2 is for smaller image sizes and it is the magnified portion of the corresponding parts in Figure 6.3. Execution time versus image size graphs for each image is given in Appendix E separately for a more detailed investigation. In the rest of the thesis, a preceding graph is usually the magnified portion of the corresponding parts in the successive plot.

RSST and Modified Fast RSST algorithms have the slowest execution time characteristic. Heap structure, which is used in the RSST algorithm for sorting the links are removed and a stack structure is introduced in Fast RSST. Finding the least weighted link process can now be done in constant time. Also

link weight updating procedure is postponed until a certain time. The speed-up of FRSST results from these changes. However Modified Fast RSST algorithm which is implemented for improving the SNR of the Fast RSST segmented image showed similar execution time characteristics with conventional RSST. In this interpretation of the algorithm link weight stacks are rebuilt after every merging process which is a time consuming job however in the original Fast RSST updating link weight stacks are rare and done when the working stack becomes empty. Due to this modification, speed up is decreased and processing time converges to the conventional RSST algorithm.

Modified RSST (MRSST) showed quite good performance. Speed up is nearly %50. MRSST algorithm uses the same structure as the original algorithm. Only a simple modification is applied. In this algorithm link cost updating is done if the change in the cost function is greater than a certain value. This value is chosen as 10% after some experimentation. With this modification, heap reconstruction is done rarely in comparison with the original algorithm and M-RSST runs faster because of this change. Experimental results show that if less then 10% changes in the link weight function are ignored and such links are not relocated, the segmentation gives reasonably good results. This is verified by examining the SNR of the image obtained by the modified RSST. SNR of the resulting image is very similar to the RSST segmented image.

Optimum Cost RSST showed the best performance regarding its running time. This cost favors the merging of small regions to bigger ones. This type of region growing pattern decreases the duplicated link removal complexity. Detailed analysis about duplicated link removal can be found in Appendix E. Speed up is very high and nearly %75.

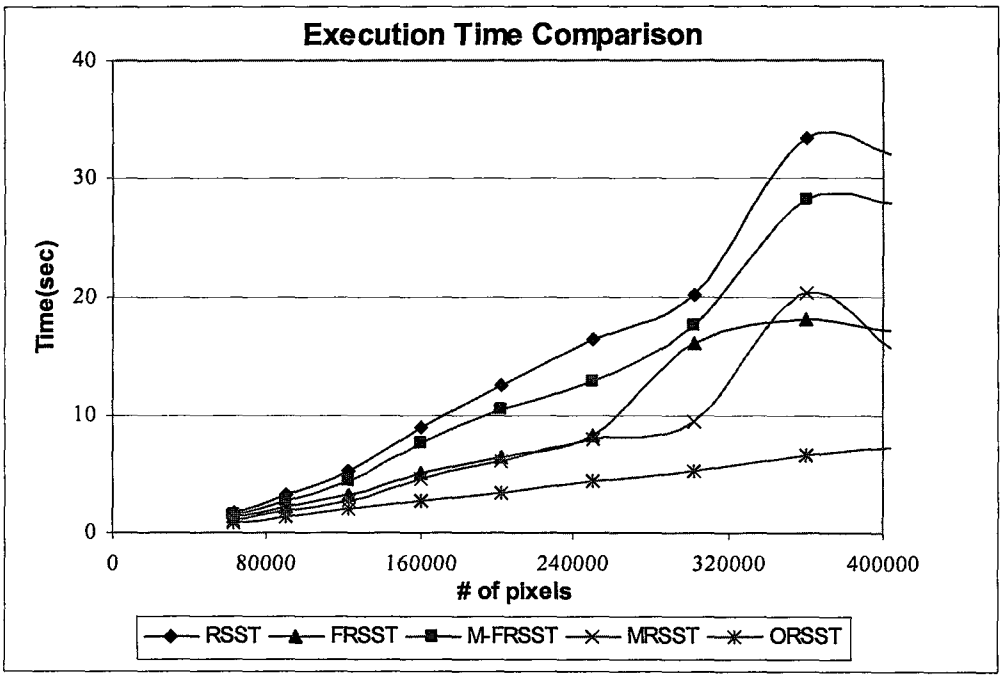


Figure 6.2. Execution time comparison between RSST, FRSST, M-FRSST, MRSST and ORSST; image size is up to 400,000 pixels, # of regions=1

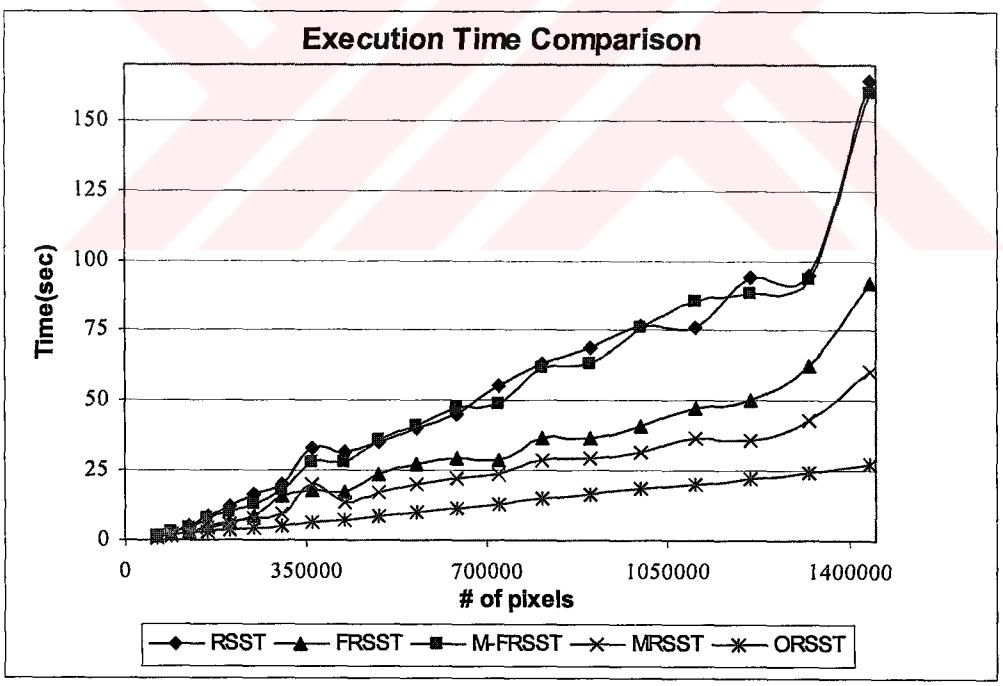


Figure 6.3. Execution time comparison between RSST, FRSST, M-FRSST, MRSST and ORSST; image size is up to 1,440,000 pixels, # of regions=1.

Results have shown that run-time performances of five of the algorithms depend on the number of pixels in the image. In execution time graphs of RSST, FRSST, M-FRSST and MRSST, time is not a linear function of the number of pixels, however, Optimum Cost RSST showed quite linear characteristics up to the tested image sizes.

6.1.2. SNR COMPARISON

It is common practice in the RSST literature to use the Signal to Noise Ratio (SNR) to evaluate the objective quality of the segmented images. SNR is defined as follows:

$$SNR = 10 \log_{10}\left(\frac{255^2}{\sigma^2_{error}}\right) \quad \text{..... (6.1)}$$

where

$$\sigma^2_{error} = \frac{1}{m\,x\,n}\sum_{i=1}^{n}\sum_{j=1}^{m}(g_{ij} - \hat{g}_{ij})^2 \quad \text{..... (6.2)}$$

The symbols g_{ij} and $\hat{g}_{ij}$ refer to the pixel intensity values of the original and the segmented image, respectively. Segmented image is expected to be similar to the original image in visual sense, so equation 6.2, which is the average mean square error, is expected to be close to zero for good segmentation. The term *mxn* refers to the size of the image and i,j refers to the indices of the image array. For this experiment, two different image sizes are selected. These are 600x600 and 900x900. Images are divided up to 10,000 regions. SNR versus image size (in pixels) graphs are shown in Figure 6.4 [image size 600x600 and region number up to 1000], Figure 6.5 [image size 600x600 and region number up to 10,000], Figure 6.6 [image size 900x900 and region number up to 1,000], and Figure 6.7 [image size 900x900 and region number up to 10,000].

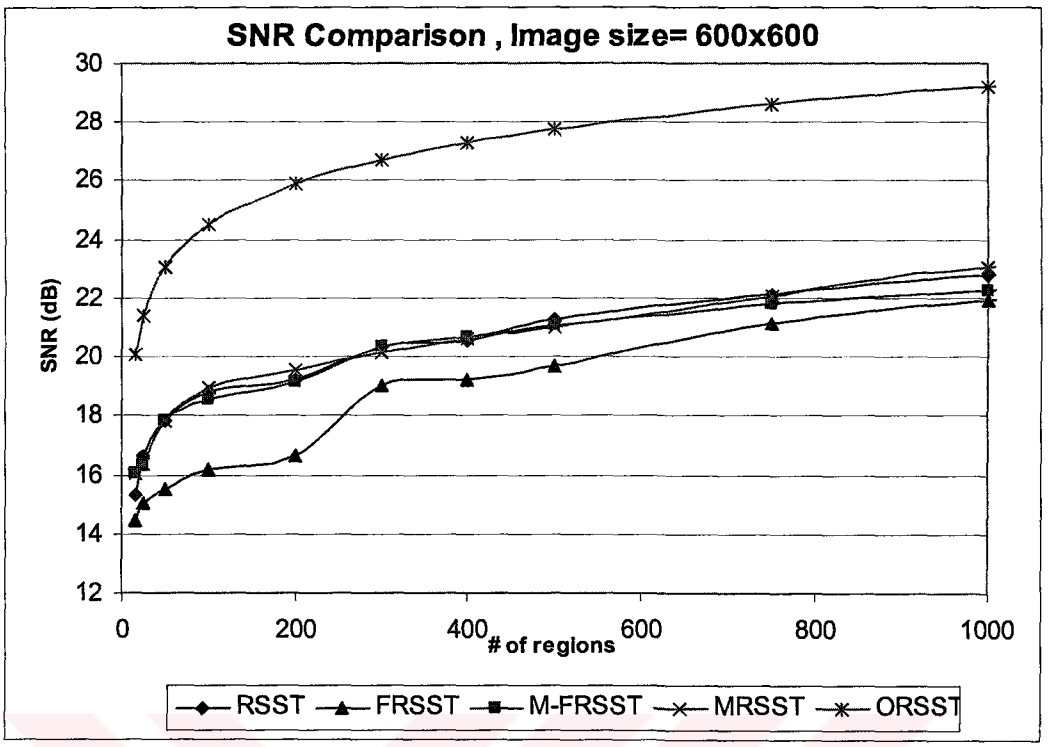


Figure6.4. SNR comparison between RSST, FRSST, M-FRSST, MRSST and ORSST, region number is up to 1000, image size is 600x600.

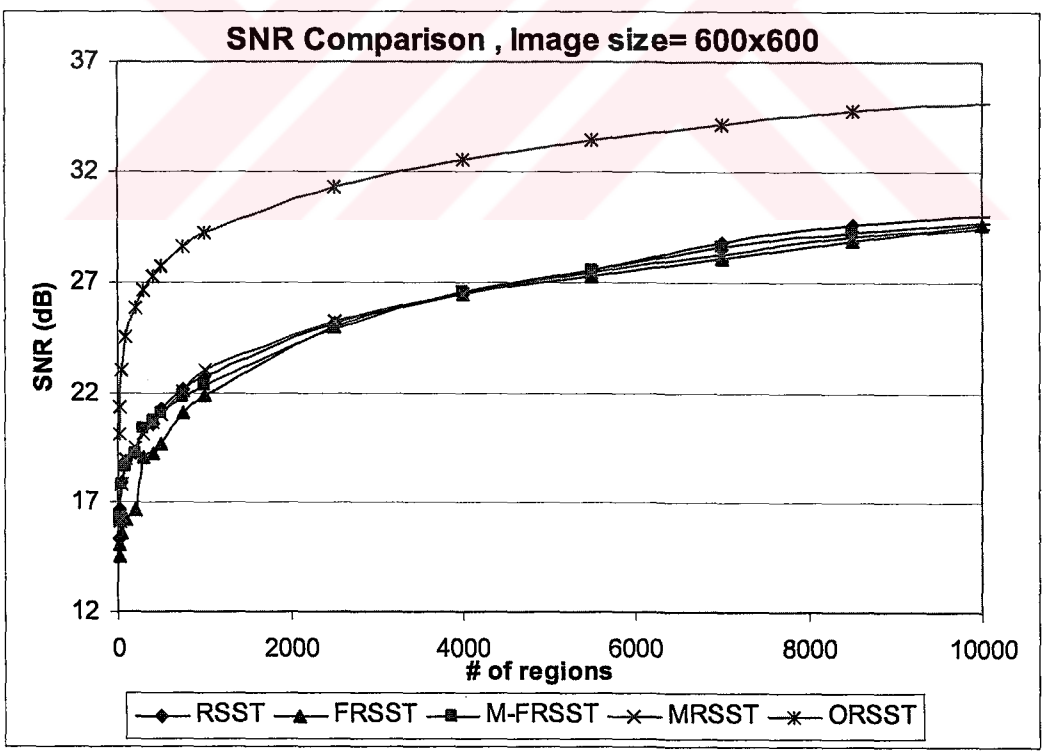


Figure6.5. SNR comparison between RSST, FRSST, M-FRSST, MRSST and ORSST, region number is up to 10,000, image size is 600x600.

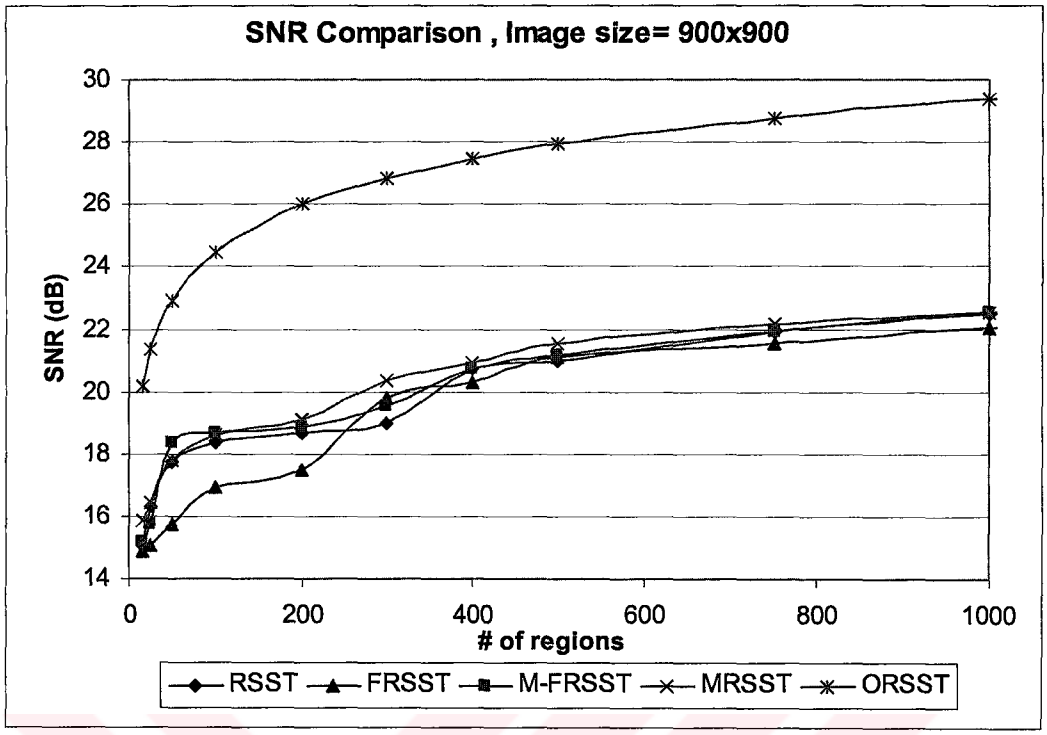


Figure 6.6. SNR comparison between RSST, FRSST, M-FRSST, MRSST and ORSST, region number is up to 1000, image size is 900x900.

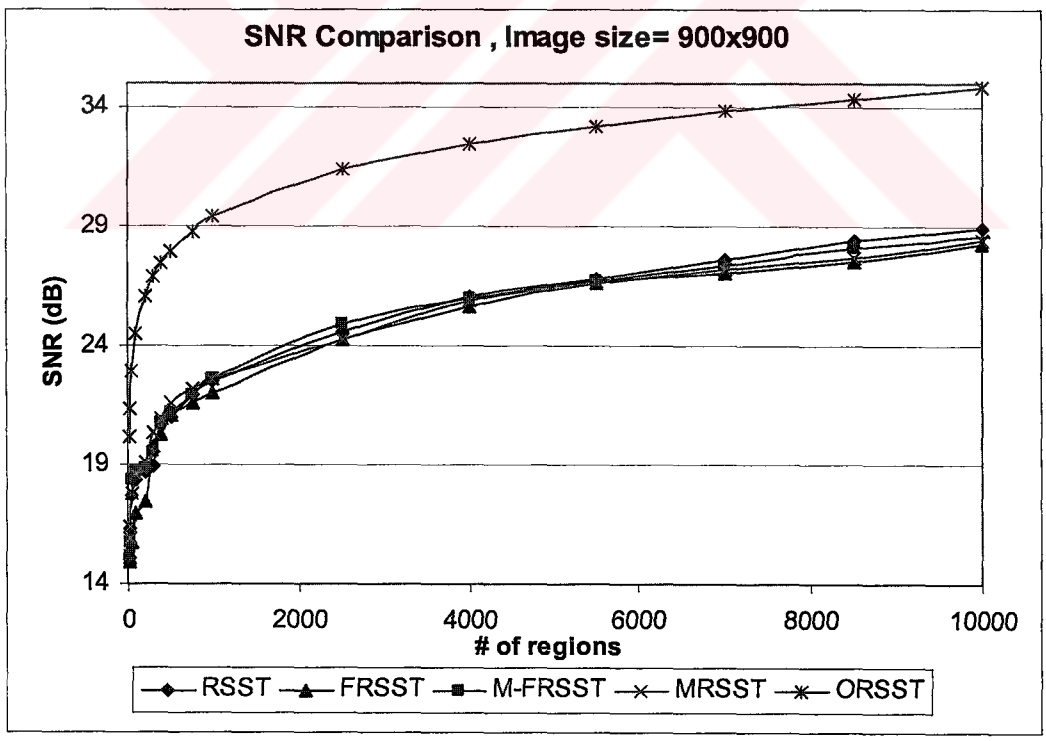


Figure 6.7. SNR comparison between RSST, FRSST, M-FRSST, MRSST and ORSST, region number is up to 10,000, image size is 900x900.

SNR graphs of the RSST, FRSST, M-FRSST and MRSST showed similar characteristics. However, SNR measure of the Optimum Cost RSST algorithm is much higher than other algorithms for each case. SNR measure of all algorithms increases as the number of regions to be segmented increases. This result is reasonable because as the number of regions to be segmented is increased pixels get closer to their original values and in the limit region number is equal to the number of pixels in the image which converges to the original image. Subjective comparisons on visual segmentation results can also be made. The segmented images for all tests for the two different sizes considered are given in Appendix F for this purpose.

6.2. COMPARISON OF SEQUENTIAL AND DISTRIBUTED ALGORITHMS

This section presents the comparison of the performances of the ORSST, and distributed RSST algorithms. In previous section it is shown that ORSST algorithm has the best performance in both execution time and SNR comparison. So now, it will make sense if we can further speed up this algorithm and therefore we compare the distributed implementation with the ORSST. In order to compare their performances, the same two criteria, namely execution time of the algorithms and objective quality of the segmented images (Signal to Noise Ratio (SNR)) are considered.

6.2.1. EXECUTION TIME COMPARISON

The aim of these experiments is to compare the run-time performance of the algorithms with respect to increasing image size. In these experiments the previously used set of images are used. Each image is resampled at 10 different sizes in this case and has a maximum size of 1,440,000 pixels (1200x1200). The

test images were already shown in Figure6.1 a,b,c,d, and e. The number of regions to be segmented is also selected to be one. Execution time of the corresponding image size is given as the average of the execution times of these five images. Same convention as in sequential algorithm comparison is applied here. For the distributed algorithm running times also include the communication overheads, i.e., the time that it takes for the slaves to send their results and for the master to receives them. Figure 4.16 and 4.17 show the execution time vs image size graphs. Number of processor is abbreviated as 'p'.

Sequential algorithm, ORSST, is run on Intel Pentium 4, 1.8GHz processor and 512 Mbytes RAM. For parallel implementation up to 4 processors are used. Two of them are Intel Pentium 4, 1.8GHz processor, and 512 Mbytes RAM, one of them is Intel Pentium 4, 2,4 GHz processor, and 256 Mbytes RAM the other one is Intel Pentium 4, 2,7 GHz processor and 512 Mbytes RAM.

Execution time improves for two and three processor cases, however for the four processors case no further improvement is observed. It is conjectured that this is due to the communication overhead increase. To investigate this, we compared the three and the four processor cases. A comparison on their average master and slave times only is carried out. In other words, the communication overhead, which is not possible to determine in the MPI environment is determined indirectly. Average slave and master time summation comparison graphs are shown in Figure 6.10. The sum of the master and slave processing times turned out to be shorter in the four processor case than the three processor one, meaning that the degradation is due to the communication overhead. Communication overhead seems to increase as the number of processors increases because of using a point to point and blocking communication structure due to the limitations of the MPI. Such a communication structure obviously increases the idle waiting times of the processors. Therefore, in a simple LAN environment similar to the one we use during these experiments, it is not advisable to use the four processor structure. On the other hand, different

architectures with better communications facilities might generate much better performance curves.

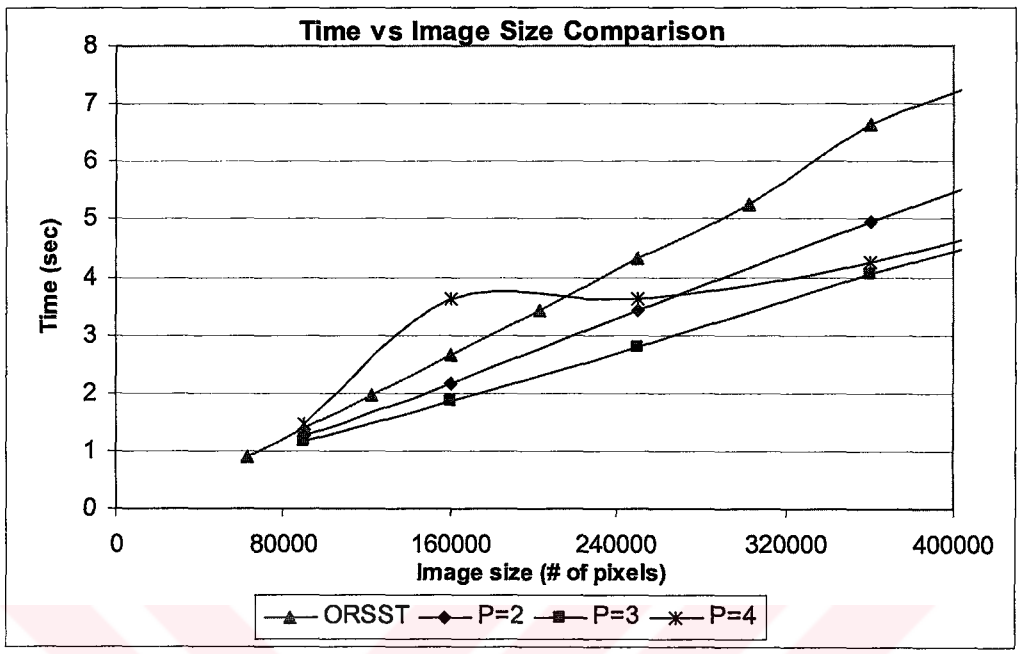


Figure 6.8. Execution time comparison of ORSST with Distributed RSST for p=2, p=3 and p=4, image size is up to 400,000, the number of regions is one.

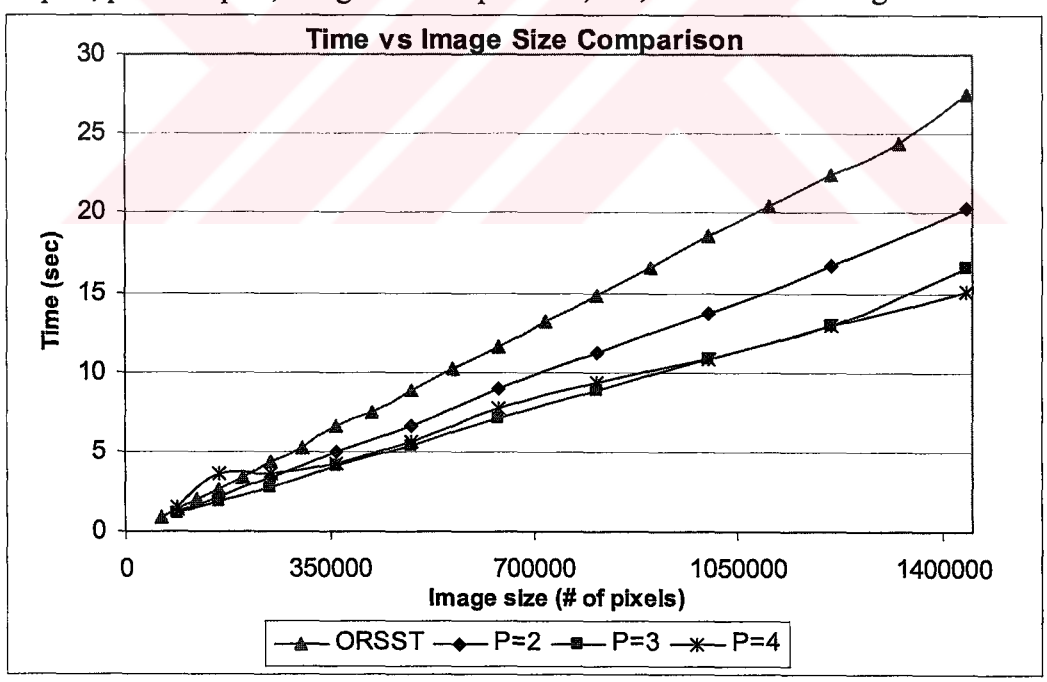


Figure 6.9. Execution time comparison of RSST with Distributed RSST for p=2, p=3, p=4, image size is up to 1,440,000 pixels and the number of region is one.

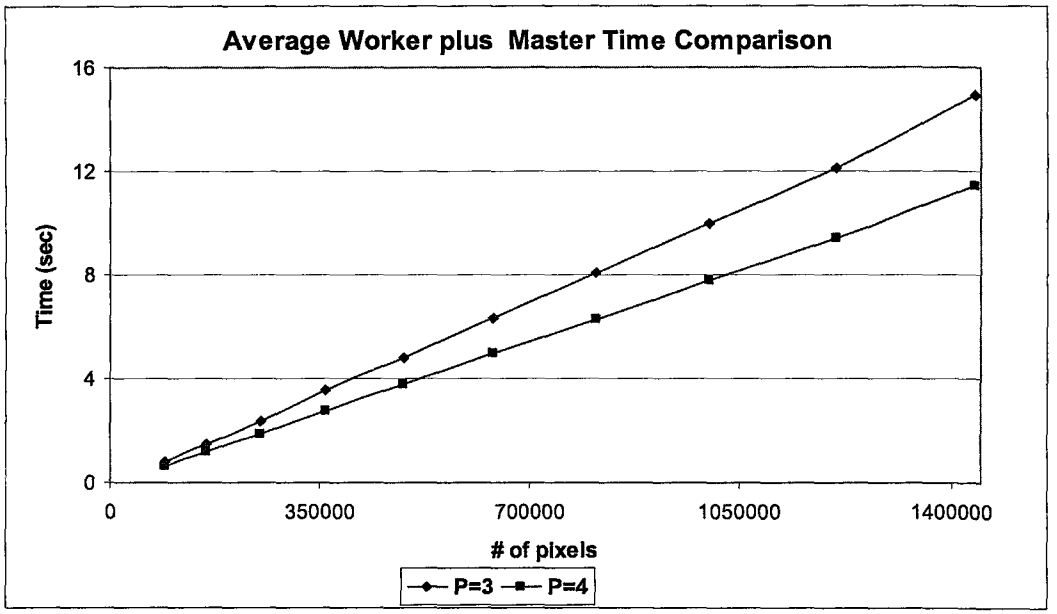


Figure 6.10. Execution time (excluding the communication times) comparison of p=3 and p=4.

6.2.2. SNR COMPARISON

It was already stated earlier that it is the common practice in the RSST literature to use the Signal to Noise Ratio (SNR) to evaluate the objective quality of the segmented images. For this comparison of the ORSST and the distributed implementation, two different image sizes are selected as 600x600 and 900x900. Images are divided up to 10,000 regions as before. SNR versus image size (in pixels) graphs are shown in Figure 6.11 [image size 600x600 and region number up to 1000], Figure 6.12 [image size 600x600 and region number up to 10,000], Figure 6.13 [image size 900x900 and region number up to 1,000], and Figure 6.14 [image size 900x900 and region number up to 10,000].

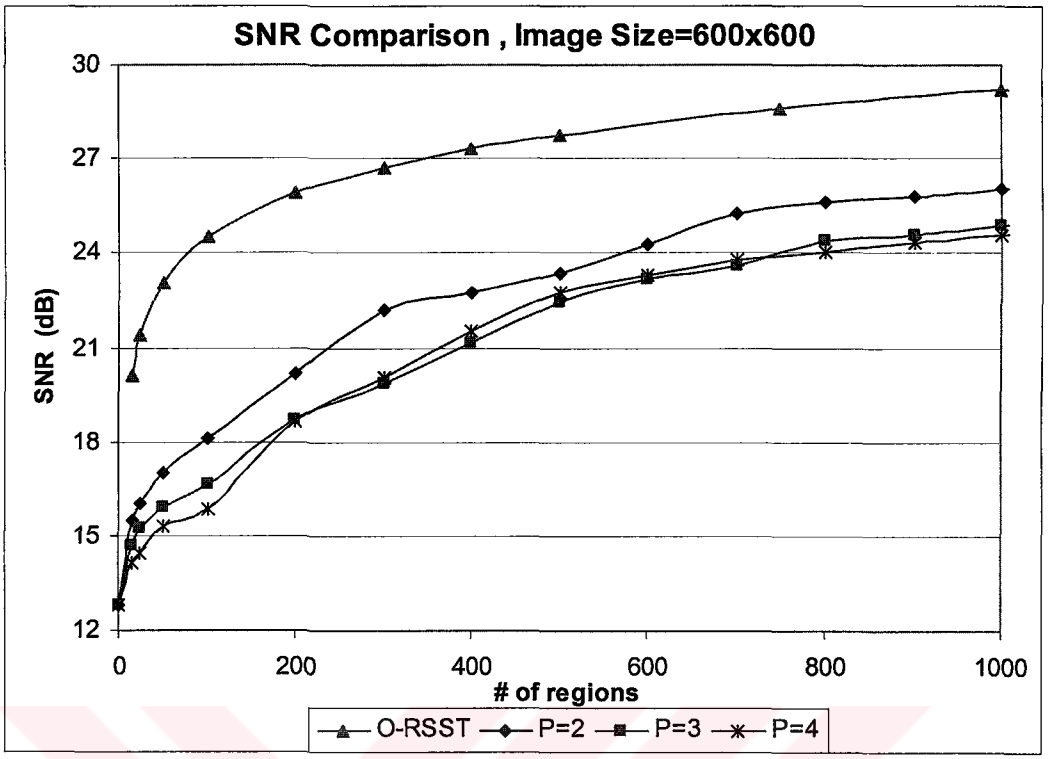


Figure 6.11. SNR comparison of distributed algorithm, region number is up to 1000, image size is 600x600.

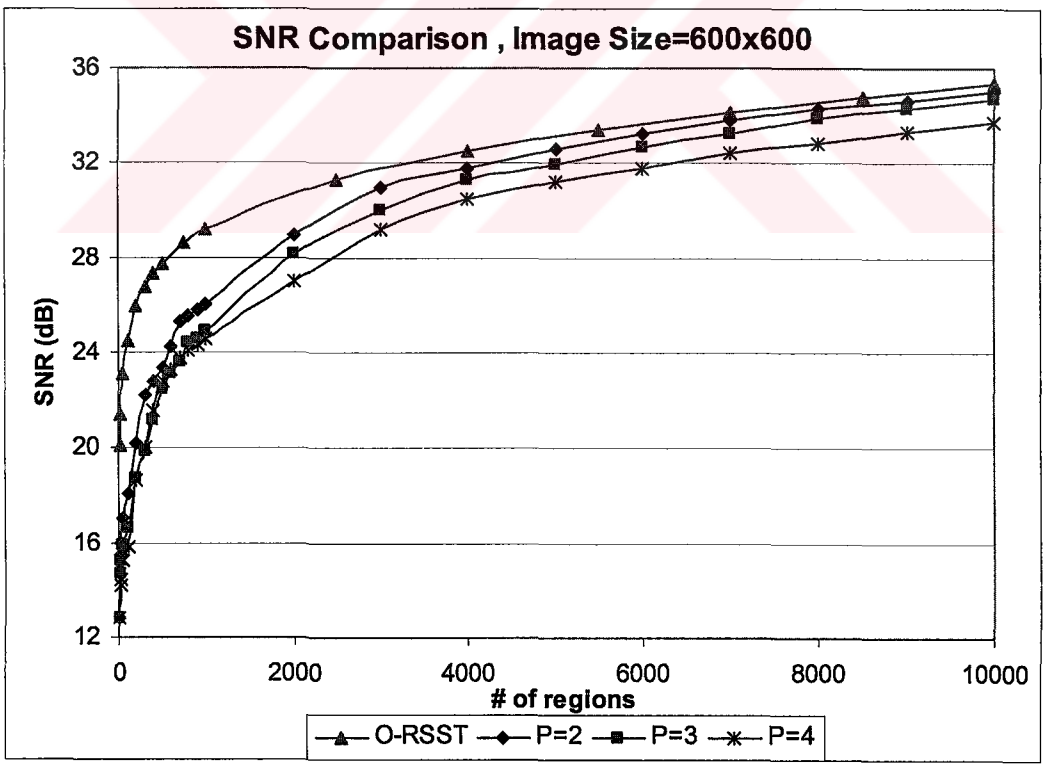


Figure 6.12. SNR comparison of distributed algorithm, region number is up to 10,000, image size is 600x600.

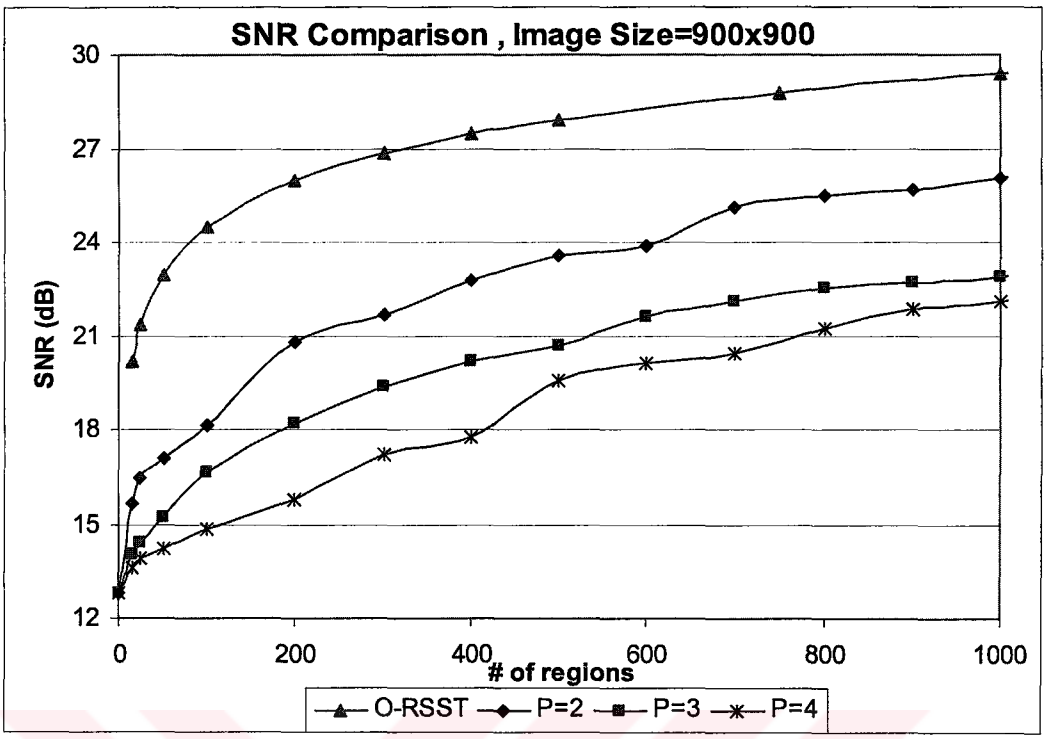


Figure 6.13. SNR comparison of sequential and distributed algorithm, region number is up to 1000, image size is 900x900.

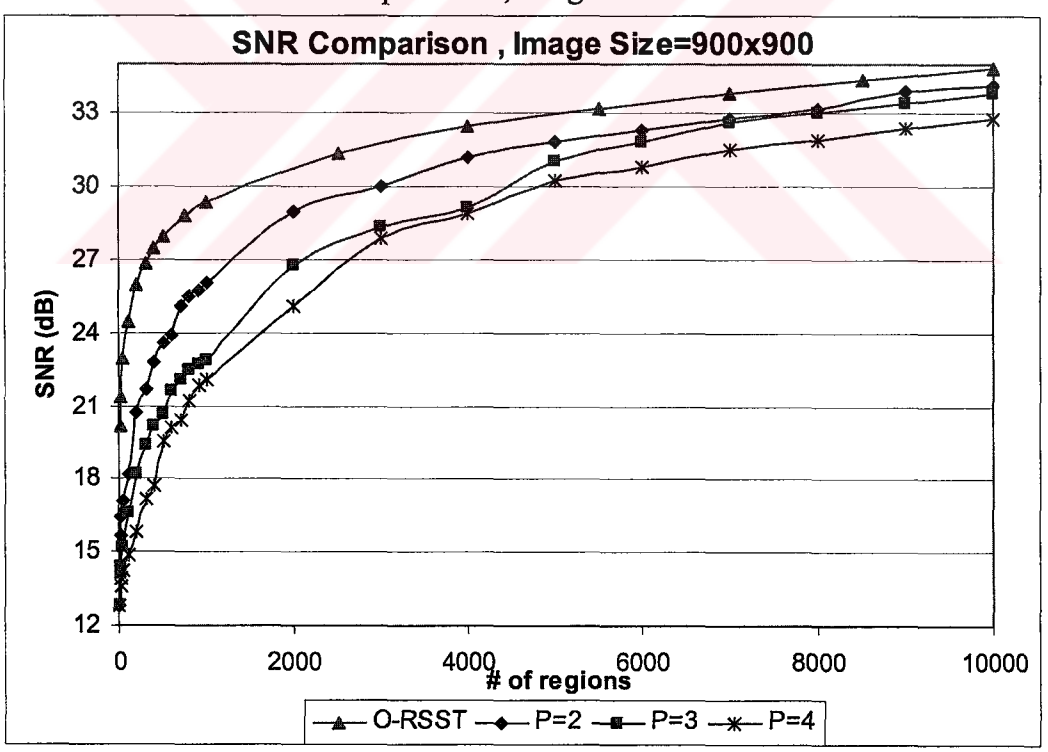


Figure 6.14. SNR comparison of sequential and distributed algorithm, region number is up to 10,000, image size is 900x900.

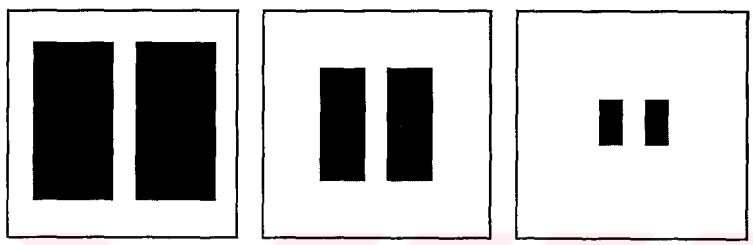
Since the global information is lost in the distributed algorithm, the SNR of the sequential ORSST turned out to be better than the distributed algorithm as expected for any number of processors. Growing pattern of the whole spanning tree radically changes because when the slaves reach their boundaries, they are forced to grow their spanning tree within their region only. As the recursive shortest spanning tree of the whole image is changed, region representations are also expected to differ. Degradation in the SNR for the distributed implementation can be acceptable for applications where speed is much more important than the high segmentation quality. The segmented images for all tests for the two different sizes considered are given in Appendix F for this purpose.

6.3. PROS AND CONS OF THE RSST ALGORITHM

The execution time of the RSST does not depend only on the image size but also on some image characteristics. It is stated in the literature that the execution time of the algorithm heavily depends on the link sorting part [7], however this may not be true for all cases. Execution times of the duplicated link removal and link cost updating operations depend on the region that is growing because at every growing step all links neighboring to that region are traversed for the number of links in the added region. As the region gets bigger, the number of neighboring links also increases. If the image contains large size objects or background, the processing time is bound by the duplicated link removal and link cost updating processes since the number of traversed links has a cumulative behavior. To validate this claim, fixed size images having different background and object sizes have been compared in the following experiment.

The aim of this experiment is to compare the running times of the ORSST algorithm for different region and background sizes for a fixed image size. Table 6.1. shows the simulation results. Image size is equal to 600x600 (width x height), however rectangular objects have different sizes in each image. As a

result, background sizes are different. The image shown in Figure 6.15.a has the smallest background and the biggest rectangle size, the image in Figure 6.15.c has the biggest background and the smallest rectangle size. Execution time of the third image is higher than the other two, since duplicated link removal process for background costs much higher. Execution time of the first image is the smallest since the total number of traversed links for duplicated link removal and also for link cost updating is less. Proof of this reasoning can be found in Appendix D.



a) b) c)

Figure 6.15. Test images with variable background sizes

Table 6.1. Comparison of execution times for variable background sizes

Image	Object Size (% of image size)	Background Size (% of image size)	Execution Time (sec) (ORSST)
Figure6.15.a	24,5%+24,5%=49%	51%	30.10
Figure6.15.b	10%+10%=20%	80%	109.17
Figure6.15.c	2%+2%=4%	96%	201.68

Another disadvantage of the RSST algorithm is its dependence on global information. The algorithm is not very suitable for simple parallelization by directly dividing the image into pieces and than merging the resulting spanning trees by a master processor. The problem has two aspects. First one is the quality

of segmentation result. The image partitions, which are assigned to different processors, are hindered to reach to other partitions. In the distributed implementation developed in this work, the heap reconstruction is done in parallel gaining valuable time but duplicated link removal, which is the duty of the master processor, is done sequentially. When duplicated link removal processing time becomes dominant in the total process, increasing the number of processors does not improve the total processing time significantly.

RSST is flexible in using different link costs. The measure of the success of segmentation result is evaluated using the "Mean Square Error" which can be minimized by the RSST using an appropriate cost function (Equation 4.4.). Therefore, if MSE minimization is considered, RSST becomes the best segmentation algorithm.

RSST is very simple because of using only gray level values of the pixels and not requiring any other information. Also the user has the freedom to decide on the number of regions to be segmented but of course determining the number of regions in automatic recognition processes remains a difficult task and is an active research area.

CHAPTER 7

CONCLUSION

A fundamental problem in image analysis is image segmentation. Since objects and background determination is done by segmentation, it has an important role also in pattern recognition. The recursive shortest spanning tree (RSST) was proven to be highly accurate to define regions. In this thesis, Recursive Shortest Spanning Tree (RSST) for image segmentation is studied. Aim of the study was to accelerate the execution time of the RSST algorithm, which might have positive effects on some applications such as video coding.

In this study, first, the conventional Recursive Shortest Spanning Tree Algorithm is reviewed and its algorithmic steps are investigated. It is implemented and tested on five different images, which are also rescaled to obtain images of different sizes. It is observed that the execution times of the RSST for five different images having the same image size are quite different. Execution time of the conventional algorithm does not only depend on the image size but also on some image characteristics such as the object size versus

background size, etc. An execution time of 371 seconds is observed for a 1200x1200 image (Image No: 4) in the experiments, which could be regarded as considerably high.

The above level of run-time performance makes it viable to search for faster implementations of the RSST. Therefore, fast RSST Algorithm (FRSST) found in the literature is reviewed and discussed in Chapter 4. It is claimed in [7] that FRSST brings a considerable improvement on RSST, however this claim was supported using only a limited size and limited number of images. Through a more comprehensive experimentation, it is observed within this study that the expected improvement on the run-time performance of the FRSST is not as much as stated. Examining the execution time and Signal to Noise Ratio (SNR) graphs, it is possible to infer that the run-time performance is improved despite a degradation in SNR figures. Fast RSST achieves this speed up by implementing a different link sorting structure, which is faster than the rearrangement of the heap structure. Subjective comparisons on visual segmentation results of the Fast RSST also indicate a poorer performance when compared to conventional RSST. This decrease in the quality is expected because link weight costs are rounded and the link weight updating process is done much rarely.

Our aim is to improve the run-time performance of the RSST preserving the quality of the segmentation but it is observed that the Fast RSST degrades the segmentation quality. To compensate for this degradation, we propose to update the link weights after every merging process in the Fast RSST, which is what the conventional RSST does. With this modification, the only difference between the modified Fast RSST (M-FRRST) and the RSST becomes the link sorting process. The RSST sorts the links using a heap data structure while M-FRRST sorting the links with rounded weights using a stack data structure. Segmentation qualities of both algorithms are shown to be similar in our computational study and no significant difference is observed in the subjective comparisons of their visual outputs. However, it is observed that the run-time performance gain in FRSST is

lost with the proposed modification. The execution times of M-FRSST are slightly better than the conventional RSST for small size images and slightly worse for larger sizes. As a result, it is possible to conclude that none of these two algorithms, RSST and M-FRSST, is superior to each other and FRSST may be used only if the speed-up is more important than the segmentation quality degradation.

Heap reconstruction in the conventional algorithm is a time consuming task and hence the studies have focused on speeding up this process. This thesis work also focuses on that aspect and recommends using the heap reconstruction selectively. Heap is a data structure that is needed to sort the links after every merging step due to the dynamically changing link weight costs. However, not every change in the link weight cost is important. Some of the link costs may change dramatically during the merging process and this change may affect the growing pattern of the spanning tree. However, some of the link costs may not alter significantly and is not that critical. For these cases where cost change may be ignored, there is no need to perform any heap operations. The thesis work have found through experimentation that ignoring heap operations for link cost changes below 10% results in similar SNR figures with original but expedites the algorithm execution time in the order of 50%. This modification is verified to bring no significant segmentation quality degradation by comparing the segmentation result of the modification and the original approach. In fact, considering the timing and SNR curves simultaneously, it can be concluded that the MRSST achieves better performance than the FRSST.

In the RSST, FRSST and simple modification work, absolute intensity difference between the regions are used as the link weight cost function. It is also compulsory to use the absolute intensity difference cost function in the fast algorithm because stacks should be finite in number. In clustering theory, a distance function, which minimizes the mean square error of the clustering, is already proposed. Minimizing MSE of a clustering, in our case segmentation,

means obtaining the best quality segmentation results. The thesis studies effects of using the alternative cost structure and finds out besides SNR improvement, run-time performance of the RSST algorithm is improved dramatically. This optimum cost (ORSST) algorithm is also found to have an almost linear execution time with respect to increasing image size. Subjective comparisons on visual segmentation results are also satisfactory. Therefore, the thesis points the ORSST as the best algorithm among the studied RSST versions according to both run-time performance and SNR figures.

To achieve further improvements in the run-time performance, a distributed implementation of the algorithm is proposed and studied. Existing work on parallel RSST is a completely theoretic one and that proposal cannot be applicable in today's computer architectures. Therefore we focused on and attempted a distributed implementation of the algorithm on a simple locally networked computers based on a simple partitioning strategy. Such networks are widespread and easily available to users. The selected distributed environment in this part is MPI and the biggest disadvantage in MPI is its support for only point-to-point communications. This forced us to design the algorithm such that the overall communication requirement is kept at a minimum. Therefore the distributed the algorithm depends on a physical and equal size partitioning of the image the number of partitions being equal to the number of available processors. In this thesis work, experiments using up to four processors are carried out.

The computational results reveal that the SNR figures of our distributed implementation are above the SNR figures of the sequential algorithms studied except ORSST, for two and in some cases for three processors. The SNR figures in the two processor case is better than three processor case for most image sizes. For four processor case, the SNR figures are nearly same as the sequential algorithms other than the ORSST.

A speed up is achieved by our distributed implementation up to three processors case beyond which the communication overhead becomes dominant. Approximately 25% and 40% speed up is obtained in two and three processor cases, respectively.

To sum up; the thesis work has aimed to accelerate the RSST algorithm keeping the quality of segmentation. The best performance among all algorithms studied regarding the segmented image quality is shown by Optimum Cost RSST, i.e., by the use of the alternative link cost function. Moreover, ORSST's run-time performance is the best among the sequential algorithms. FRSST, M-FRSST and MRSST expedite the conventional RSST but their SNR figures and speed ups still lag the ORSST. Speed-up obtained by the distributed RSST by three processors is the highest among other implementations however segmentation quality still lags ORSST. For applications where the quality of segmentation is more critical, ORSST will be the appropriate solution. For applications where the run-time performance is more crucial than the high quality of the segmentation, the distributed RSST using two processor may fit the requirements.

This study accelerated the RSST algorithm mostly by concentrating on the link sorting part, however the complexity of RSST does not only depend on the image size and the number of links to be sorted but also on the object and background sizes.

As future work, alternative partitioning strategies can be investigated and the duplicated link removal process, which seems to be the most time consuming part for some of the cases, can be focused for parallelization.

.

REFERENCES

[1] S. Dickinson, M. Pelillo, R. Zabih, "Introduction to the Special Section on Graph Algorithms in Computer Vision", *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 23, no. 10, October 2001

[2] Notes on Image Segmentation, Computer Based Learning Unit, University of Leeds [1998], Retrieved June, 2004, Web Site, http://rkb.home.cern.ch/rkb/AN16pp/ node131.html

[3] A. A. Alatan (2004), Lecture Notes on Pattern Recognition, Unsupervied Learning, Retrieved Jan., 2005, Web Site, http://www.eee.metu.edu.tr/~alatan/

[4] O. J. Morris, J. M. Lee, and A. G. Constantinides, "Graph theory for image analysis: An approach based on the shortest spanning tree," *Proc. Inst. Electr. Eng. (Part F)*, vol. 133, 1986, pp. 146–152.

[5] O. J. Morris, J. M. Lee, and A. G. Constantinides, "A unified method for segmentation and edge detection using graph theory," *Proc. Int. Conf. Acoustics, Speech, and Signal Processing*, 1986, pp. 2051–2055.

[6] S.H. Kwok, A. G. Constantinides, W.C. Siu, "An Efficient Recursive Shortest Spanning Tree Algorithm Using Linking Properties", *IEEE Transactions On Circuits And Systems For Video Technology,* Vol. 14, No. 6, June 2004

[7] S. H. Kwok and A. G. Constantinides, "A fast recursive shortest spanning tree for image segmentation and edge detection," *IEEE Trans.Image Processing*, vol. 6, pp. 328–332, Feb. 1997.

[8] S. H. Kwok and A. G. Constantinides ,"A parallel recursive shortest spanning tree algorithm for image segmentation in distributed computing environment," *J. Parallel Distrib. Comput.*, vol. 56, no. 3, pp. 181–207, Mar. 1999.

[9] Y. Xu, E.C. Uberbacher, "2-D image segmentation using Minimum spanning Trees", *Oak Ridge National Laboratory Report,* September 1995.

[10] P. Feleznszwalb and D. Huttenlocher. "Image segmentation using local variation". *Proceedings of IEEE conf. Computer Vision and Pattern Recognition,* pp 98–104, 1998.

[11] C.T. Zahn. "Graph-theoretic methods for detecting and describing gestalt clusters". *IEEE Trans. Comput.*, vol 20, pp 68-86, 1971.

[12] R. Urquhart. "Graph theoretical clustering based on limited neighborhood sets". *Pattern Recognition*, vol 15:3, pp 173-187, 1982.

[13] Graph-Theoretic Approaches, *from Ben Appleton, University of Queensland*, Retrieved June, 2004, Web site, http://www.itee.uq.edu.au/~appleton/Graph.htm

[14] Y. Zeng, "Perceptual segmentation algorithm and its application to image coding," in *Proc. Int. Conf. Image Processing*, 1999, pp. 820–824.

[15] M. J. Biggar, O. J.Morris, and A. G. Constantinides, "Segmented-image coding: Performance comparison with the discrete cosine transform," in *Proc. IEE (Part F)*, vol. 135, Apr. 1988, pp. 121–132.

[16] M. J. Biggar, and A. G. Constantinides, "Segmented video coding," in *Proc. IEEE. Int. Conf. Acoustics,Speech, and Signal Processing*, 1988, pp. 1108–1111.

[17] OpenCV Library Free Source, *SourceForge*, Retrieved January,2004, SourceForge Web Site, http://sourceforge.net/projects/opencvlibrary

[18] Graph theory Tutorials from *Chris Coldwell, University of Tennessee at Martin*, Retrieved February,2004, Web Site, http://www.utm.edu/departments/math/graph/

[19] Message Passing Library Implementation Free Source *authored by Argonne National Laboratory Group and Mississippi State Group*, Retrieved October,2004, Web Site http://www-unix.mcs.anl.gov/mpi/mpich/

[20] Large Size Images from *National Aeronautics and Space Administration,* Retrieved May,2004, Web Site, http://grin.hq.nasa.gov/BROWSE/DFRC_1.html

[21] An interactive image Processing course in *Delft University of Technology,* Retrieved May, 2004, Web Site, http://www.ph.tn.tudelft.nl/Courses/FIP/

[22] O. J. Morris and A. G. Constantinides, "Progressive image coding from a

spanning tree image representation," *presented at the Onzieme Colloq., Nice, France,* 1987.

[23] S. H. Kwok and A. G. Constantinides, "A scaleable and adaptive temporal segmentation algorithm for video coding," *Graphical Models and Image Processing*, vol. 59, no. 3, pp. 128–138, 1997.

[24] S. H. Kwok, W. C. Siu, and A. G. Constantinides, "Adaptive temporal decimation algorithm with dynamic time window," *IEEE Trans. CircuitsSyst. Video Technol.*, vol. 8, pp. 104–111, Feb. 1998.

[25] A. A. Alatan, L. Onural, M. Wollborn, R. Mech, E. Tuncel, and T. Sikora, "Image sequence analysis for emerging interactive multimedia services—The European COST 211 framework," *IEEE Trans. CircuitsSyst. Video Technol.*, vol. 8, pp. 802–813, Nov. 1998. China, 1995, pp. 1386–1389.

[26] S. Cooray, N. O'Connor, S. Marlow, N. Murphy, and T. Curran, "Hierarchical semi-automatic video object segmentation for multimedia applications," in *Proc. SPIE*, vol. 4519, 2001, pp. 10–19.

[27] D. Tancharoen, S. Jitapunkul, S. Triamlumlerd, P. Kittipanya-Ngam, and S. Chompun, "Object segmentation based on multiple features for low bit rate video coding," in *Proc. 5th Int. Conf. Signal Processing and the16th World Computer Congress*, 2000, pp. 975–978.

[28] E. Tuncel and L. Onural , "Video object segmentation by extended recursive-shortest-spanning-tree method," in *Proc. IEEE-EURASIPWorkshop Nonlinear Signaland Image*, 1999, pp. 23–27.

[29] E. Tuncel and L. Onural, "Utilization of the recursive shortest spanning tree algorithm for video-object segmentation by 2-D affine motion modeling," *IEEE Trans. Circuits Syst. Video Tech.*, vol. 10, pp. 776–781,Aug. 2000

[30] A. D. Doulamis, N. Doulamis, and S.Kollas, "Non-sequential video content representation using temporal variation of feature vectors," *IEEE Trans. Consumer Electron.*, vol. 46, pp. 758–768, Aug. 2000.

[31] N. D. Doulamis, A. D. Doulamis, Y. S. Avrithis, K. S. Ntalianis, and S. D. Kollias, "Efficient summarization of stereoscopic video sequences," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 10, pp. 501–517, June 2000.

[32] Y. Avrithis, N. Tsapatsoulis, and S. Kollias, "Color-based retrieval of facial images," *presented at the 10th Eur. Signal Processing Conf. Signal Processing Theories and Applications, Tampere, Finland*, 2000.

[33] K. S. Ntalianis, N. D. Doulamis, A. D. Doulamis, and S. D. Kollias, "An active contour-based video object segmentation scheme for stereoscopic video sequences," in *Proc. 10th Mediterranean Electrotechnical Conf. Information Technology and Electrotechnology for the MediterraneanCountries*, May 2000, pp. 554–557.

[34] A. Doulamis, N. Doulamis, K. Ntalianis, and S. Kollias, "Efficient unsupervised content-based segmentation in stereoscopic video sequences," *Int. J. Artificial Intell. Tools (Architectures, Languages, Algorithms)*, vol. 9, no. 2, pp. 277–303, 2000.

[35] Y. S. Avrithis, A. D. Doulamis, N. D. Doulamis, and S. D. Kollias, "A stochastic framework for optimal key frame extraction from MPEG video databases," *Comput. Vis. Image Understanding*, vol. 75, pp. 3–24, July 1999.

[36] T. Vlachos and A. G. Constantinides, "A graph-theoretic approach to color image segmentation and contour classification," *presented at the Int. Conf. Image Processing and its Applications, Maastricht, The Netherlands*, 1992.

APPENDICES

APPENDIX A. EXPLANATIONS OF THE MAIN DATA STRUCTURES

RA: region array

An array of (rsst_region_t) elements, containing information about each region.

LA: link array

An array of (rsst_link_t) elements, containing information about each link between two regions. It is never accessed directly, but through pointers to it stored in LPA and RLL.

LPA: link pointer array

An array of integers containing pointers to the elements in LA. The array is arranged by link cost to form a heap (a balanced binary tree in which each parent has always lower cost than its two children). The heap structure means that the lowest cost link will always be the first one (note that the first element in LPA is element #1, not #0).

RLL: region link list

A linked list of elements which contains pointers to all links (i.e., LA elements) of the region.

PL: pixel list

A linked list of elements which contains the indexes of all pixels in a region."

APPENDIX B. DATA STRUCTURES USED IN COST 211 AM PROJECT

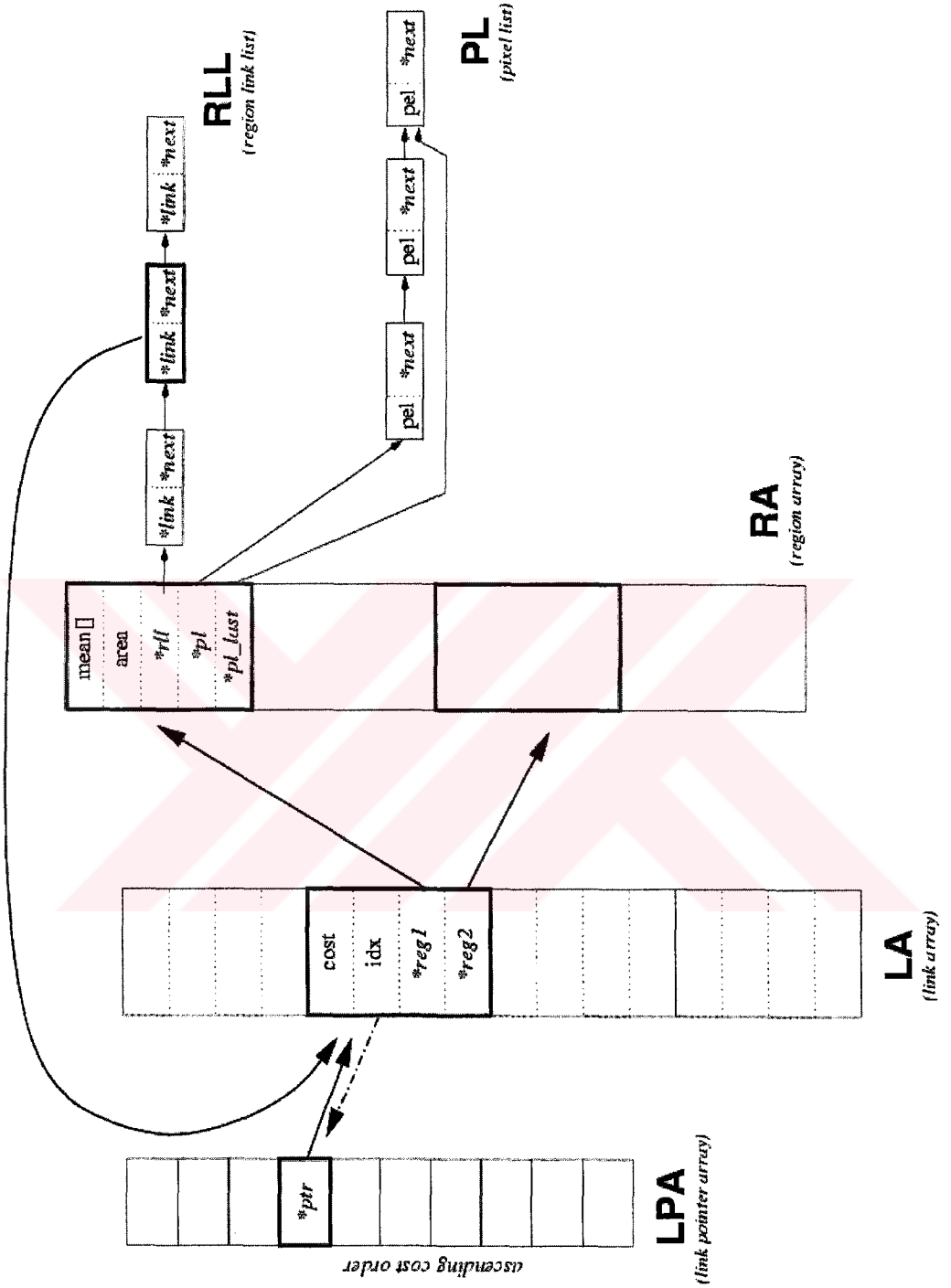


Figure A1. RSST basic data structures

APPENDIX C. RSST ALGORITHM USED IN COST 211 AM PROJECT

The following is an explanation of the three main steps:

C.1 Initialisation, in "InitColourRSST()"

Creates & fills in arrays & structures

1. Type dependent structures.Region array (LA): fills in the mean & area for every region (each region is formed by a single pel).

2. Type independent structures.

2.1 Pixel list (PL): sets the PL for every region.

2.2 LPA & RLL: creates a link between each pair of adjacent regions (i.e., adjacent pels). Each link stored in LPA must be followed by a reorder of the LPA structure (reheap).

C.2 Region merge, in "MergeRegionRSST()"

We take the first link in the LPA, which will have the lowest cost, and join its two regions.

Let's say region #B is to be added to region #A. List of operations needed:

1. UpdateRegion

1.1 Actualize #A entry in RA: compute new mean & new area of #(A+B) and store it in RA --> this operation (new mean) is different for Colour & MV

1.2 Add pixels of #B to #A. Easy, just add #B's PL to #A.

2. UpdateLinks

2.1 Update links for region #B

For each link in #B's RLL

if #A already had that link (which includes the link #A-#B) mark as deleted, remove from LPA, reheap LPA else make it point to #A add it to #A's RLL

2.2 Update links for region #A

For each link in #A's RLL

if it's marked as deleted, remove from #A's RLL, else (i.e. link to #C) compute new cost of link (cost of joining #(A+B) with #C)--> this operation is different for Colour & MV,reheap LPA

C.3) Mask creation, in "CreateMaskRSST()"

```
create a mask image ,index = 0
for each link in LIA
  for each link's region
      if area of region is nonzero
        set area of region to zero
        for each pixel in region's RLL
            set mask[pixel] = index
        index++
```

APPENDIX D. COMPLEXITY ANALYSIS OF DUPLICATED LINK REMOVAL PROCESS IN RECURSIVE SHORTEST SPANNING TREE

Assume the image being processed has a constant intensity rectangular shape shown in FigureA.2 for the simplicity of analysis. Let the width of the region be w, and the height of the region be h.

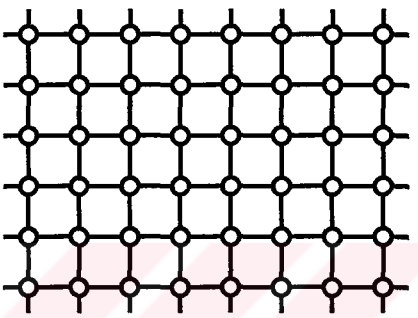


Figure A.2. Constant intensity rectangular region.

Initially each pixel in that region represents an individual region and all the links within that region has a zero cost. A worse, but not the worst, growing scenario of that region can be as follow: Each pixel at the beginning of each row grows towards to the right of itself up to the end of the row. Now each row represents a region. After that point rows will be merged to obtain a single region.

To analyze the complexity of duplicated link removal process of this scenario how many links are accessed throughout the process will be found. At every merging step number of the links being accessed equal to the product of the number of the links' of the growing region and number of the links' of added region plus number of links in the growing region.

TableA.1.Table of number of links which are processed during merging

Links incident to the growing region (Before Merging)	Number of links that are accessed in the merging	Links incident to the growed region (After merging)
4	4+4x4	20
6	6+4x6	30

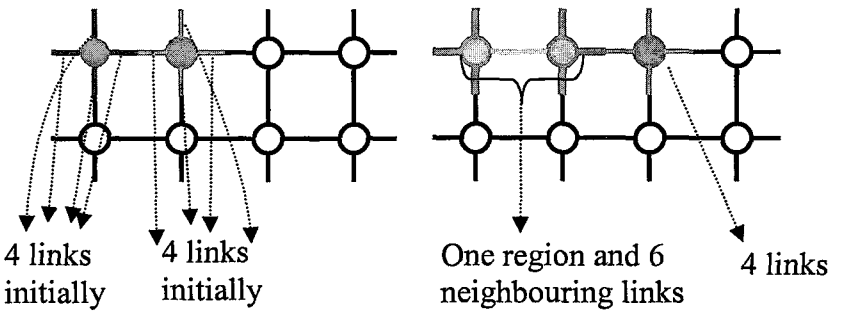


Figure A.3. Merging process and link numbers in the growing region

TableA.2.Table of number of links which are processed during merging

Links incident to the growing region (Before Merging)	Number of links that are accessed in the merging	Links incident to the growed region (After merging)
4	4+4x4	20
6	6+4x6	30
8	8+4x8	40
10	10+4x10	50
12	12+4x12	60
14	14+4x14	70

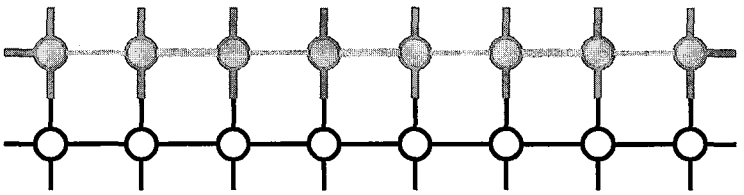
To represent one row as a single region link number that is accessed is:

$$\#\ \text{of links} = 4x2\ (2+3+..(w+1)) + 2\ (\ 2 + 3 + ...+ (w+1)\ ) \quad \text{(A.1)}$$

Region with h row height has to do this for h number of rows. So total number of links touched for inter row merging is equal to

$$\#\text{ of links} = h\,.\,[\,5w^2 + 5w] = 5w^2.h + 5.w.h \quad \text{.....(A.2)}$$

Each row represents a region and has [2.(w +1)] links neighboring to it FigureA.4.



FigureA.4. Each row represents one region

Number of links accessed in row merging:

Merging 1st and 2nd row:

links touched= [2.(w +1)] + [2.(w +1)] [2.(w +1)] $=2w^2+6w+4$

Merging new region with 3rd row:

links touched= [2.(w +1)] + [2.(w +1)] [2.(w +1)]= $2w^2+6w+4$.....(A3)

This procedure repeats until all the rows are included to the growing region. So total number of links touched for row merging for h row image is equal to:

$$\#\text{ of links} = (2w^2+6w+4)\,.\,(h-1) \quad \text{.....(A4)}$$

Total number of links touched from the beginning of the procedure is:

$$\text{Total number of links} = \underbrace{(5w^2.h + 5.w.h)}_{\text{Eqn A.2.}} + \underbrace{((2w^2+6w+4)\,.\,(h-1))}_{\text{Eqn A.4.}} \quad \text{.....A.5.}$$

$$=(7w^2h+11wh+4h-(2w^2+6w+4)) \quad \text{.....(A.5)}$$

The term "w.h" is equal to the number of pixels in that region, call it n. By examining the equation it is found that complexity of this duplicated link removal is equal to O (n.w) where n is the pixel number in that region.

APPENDIX E. TIME AND SNR GRAPHS OF IMAGES

E.1. EXECUTION TIME COMPARISON OF SEQUENTIAL ALGORITHMS

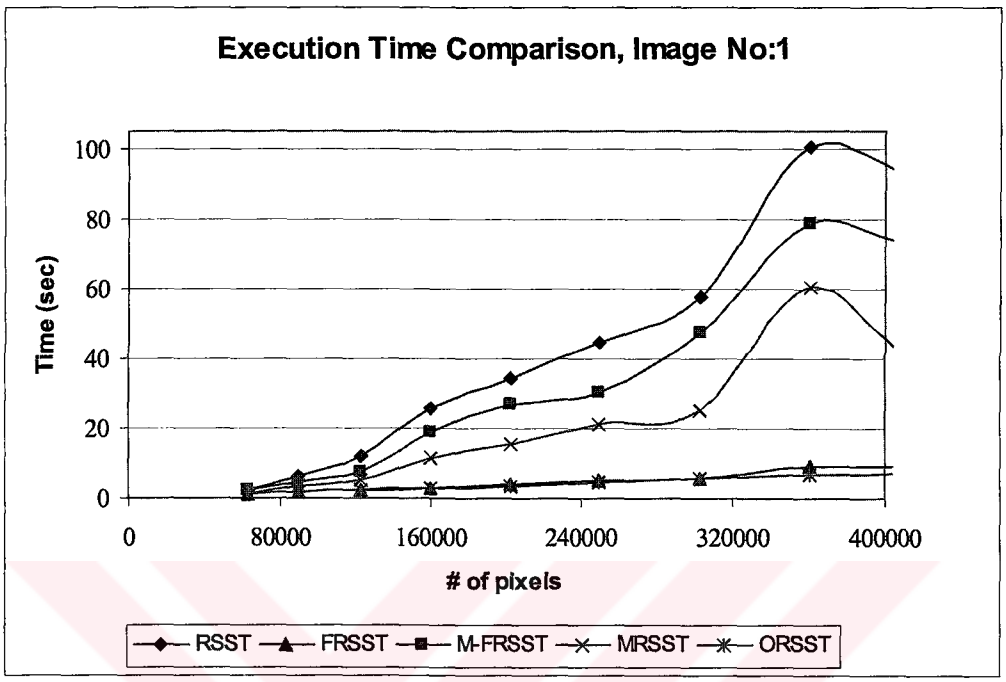


Figure A.5. Execution time comparison of image number 1, image size is up to 400,000 pixels, # of regions=1

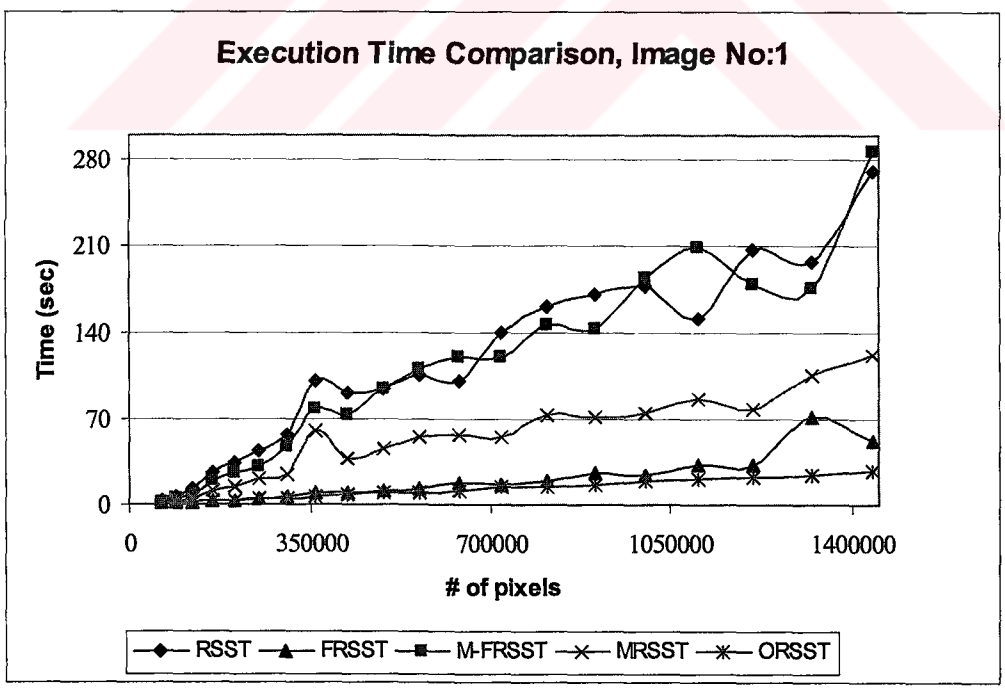


Figure A.6. Execution time comparison ofr image number 1, image size is up to 1,440,000 pixels, # of regions=1

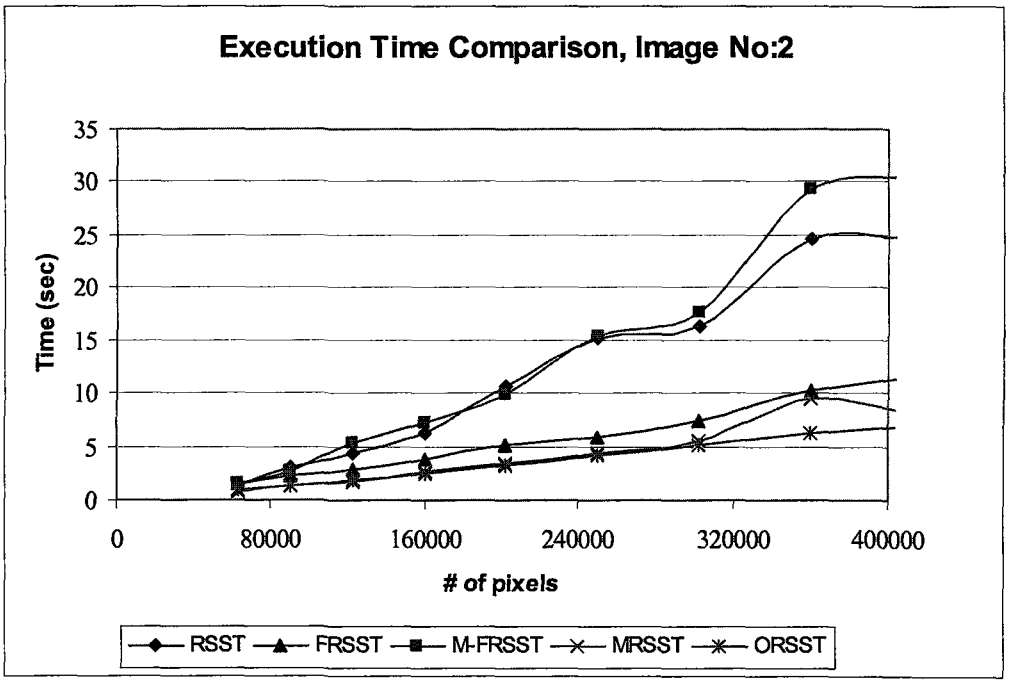


Figure A.7. Execution time comparison of image number 2, image size is up to 400,000 pixels, # of regions=1

Figure A.8. Execution time comparison of image number 2, image size is up to 1,440,000 pixels, # of regions=1

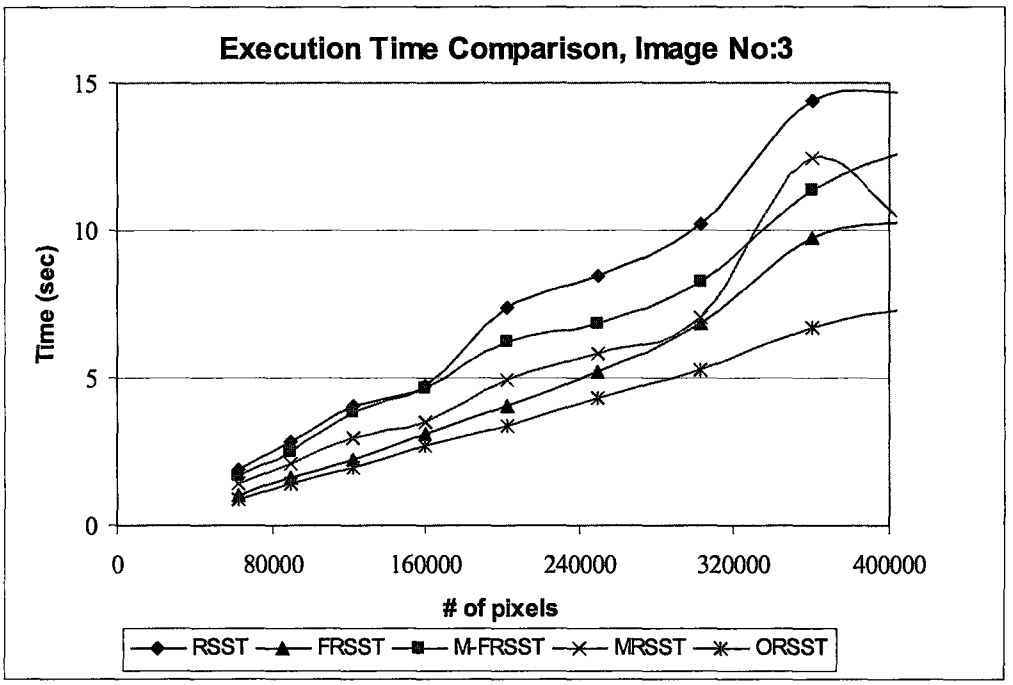


Figure A.9. Execution time comparison of image number 3, image size is up to 400,000 pixels, # of regions=1

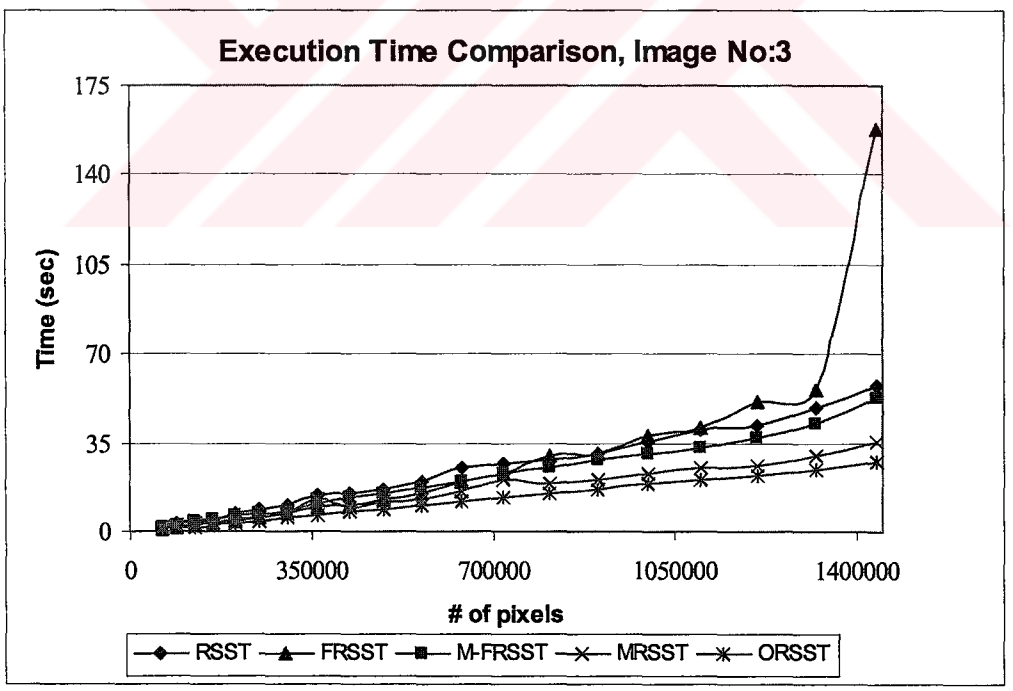


Figure A.10. Execution time comparison of image number 3, image size is up to 1,440,000 pixels, # of regions=1

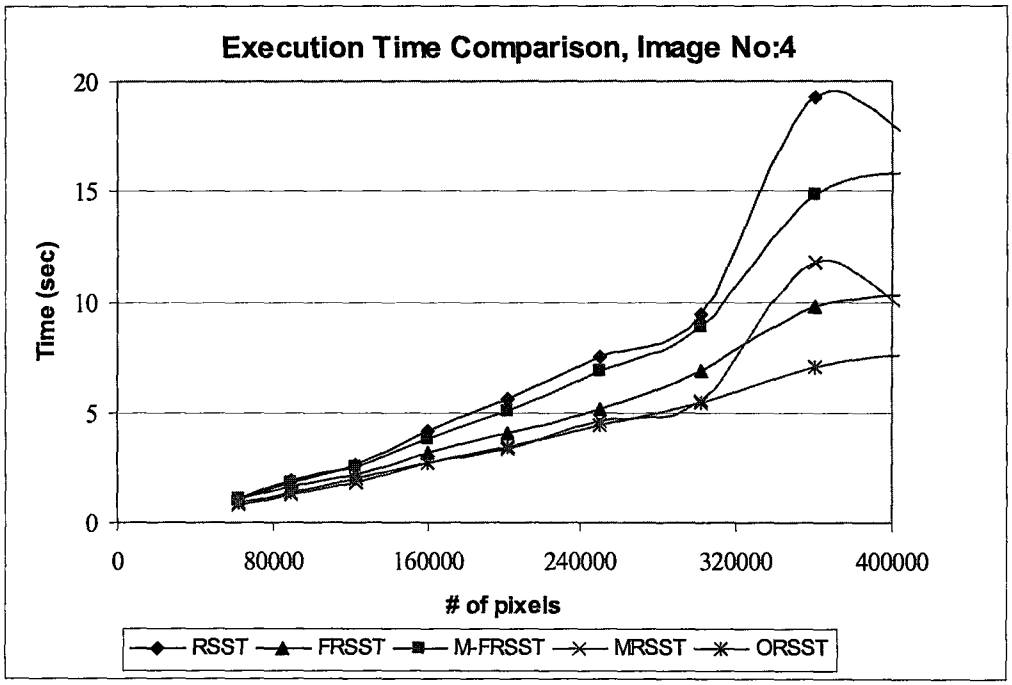


Figure A.11. Execution time comparison of image number 4, image size is up to 400,000 pixels, # of regions=1

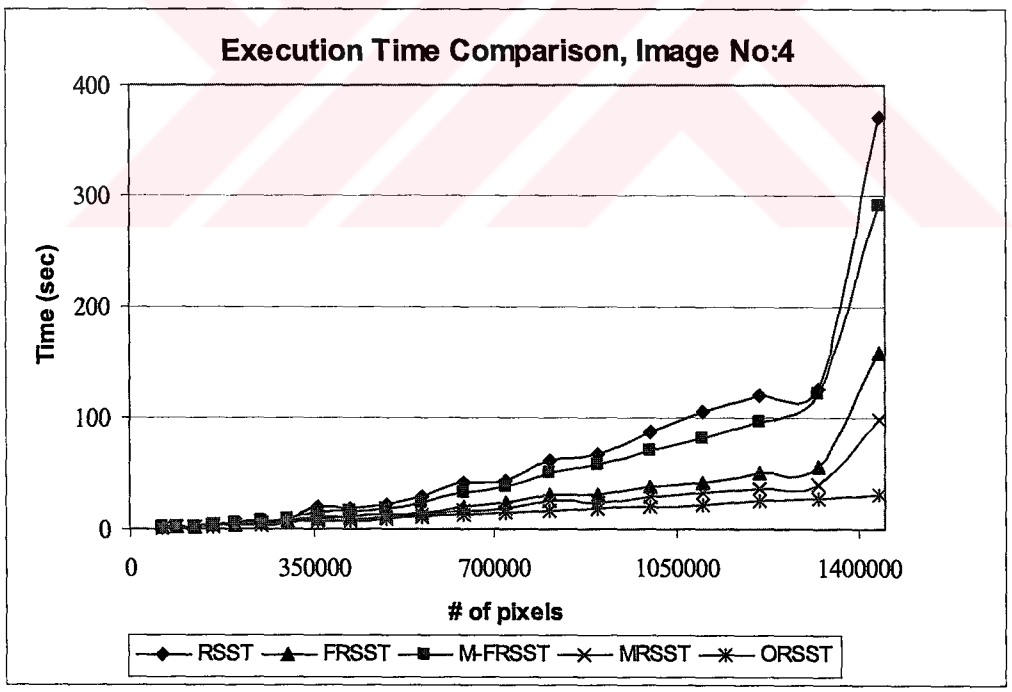


Figure A.12. Execution time comparison of image number 4, image size is up to 1,440,000 pixels, # of regions=1

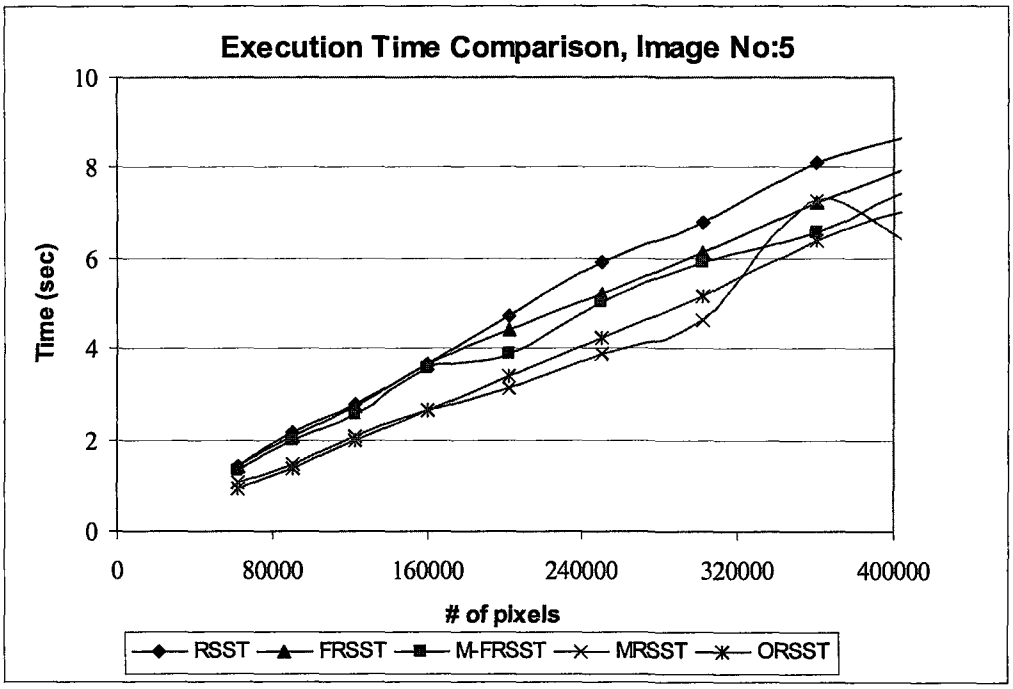


Figure A.13. Execution time comparison of image number 5, image size is up to 400,000 pixels, # of regions=1

Figure A.14. Execution time comparison of image number 5, image size is up to 1,440,000 pixels, # of regions=1

E.2. SNR COMPARISON OF SEQUENTIAL ALGORITHMS

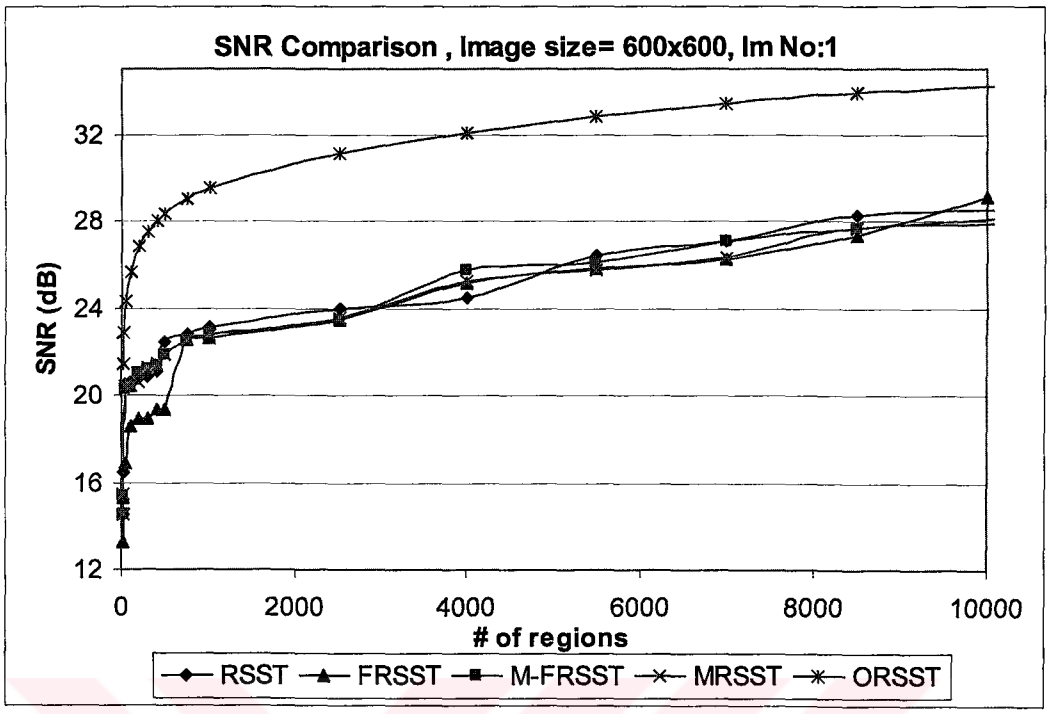


Figure A.15. SNR comparison of image number 1, region number is up to 1000, image size is 600x600.

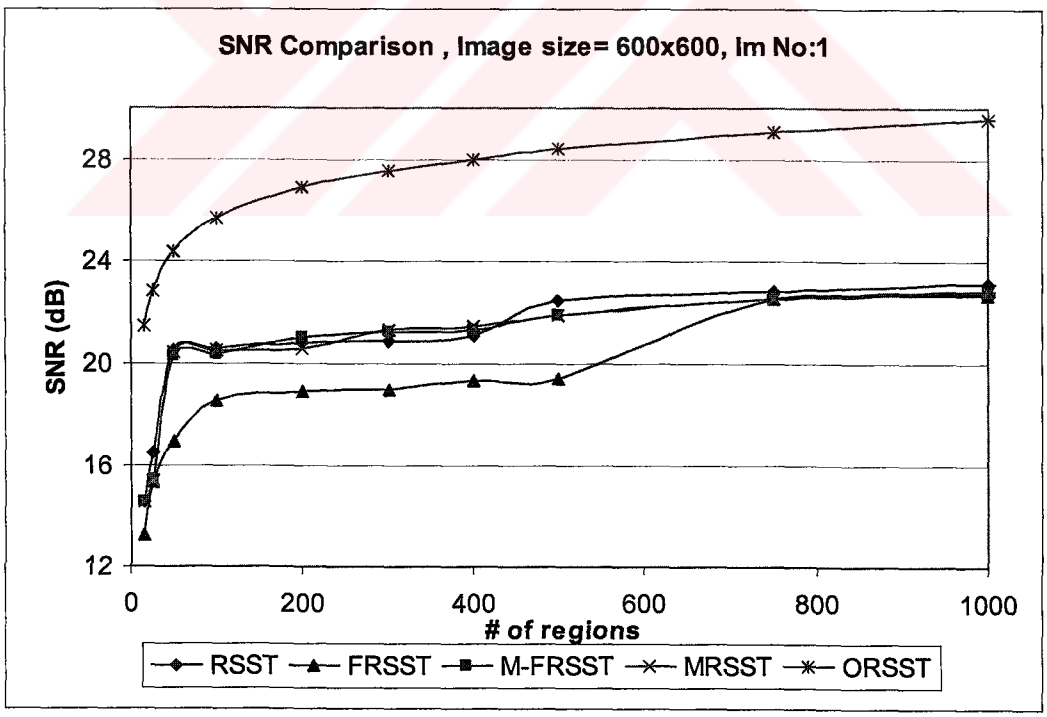


Figure A.16. SNR comparison of image number 1, region number is up to 10,000, image size is 600x600.

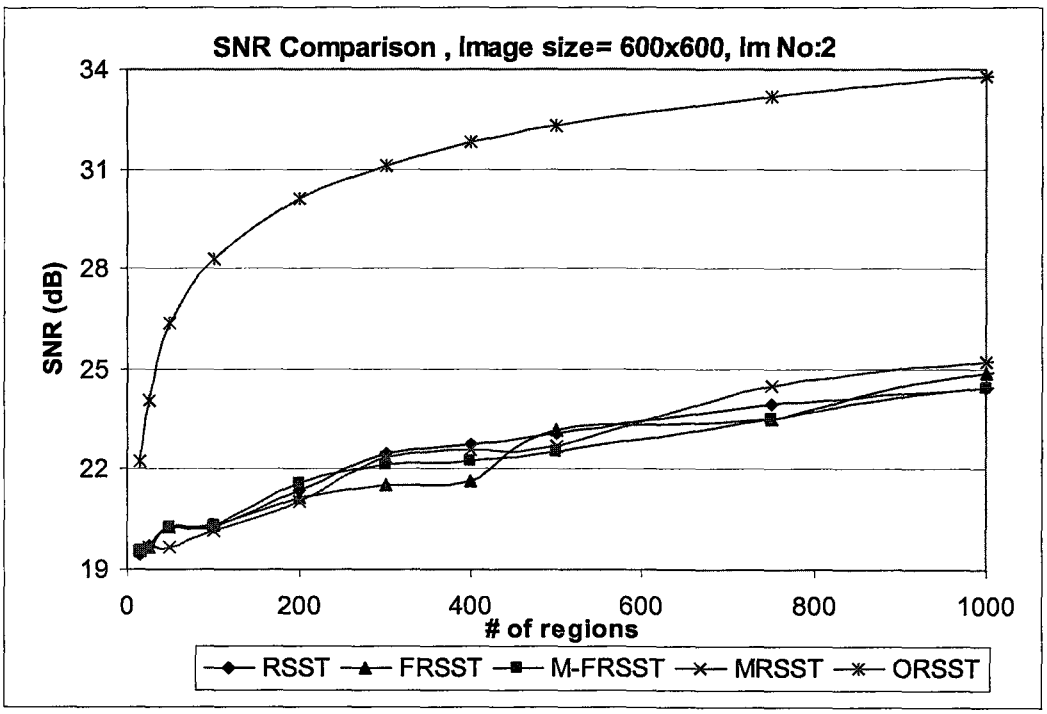


Figure A.17. SNR comparison of image number 2, region number is up to 1000, image size is 600x600.

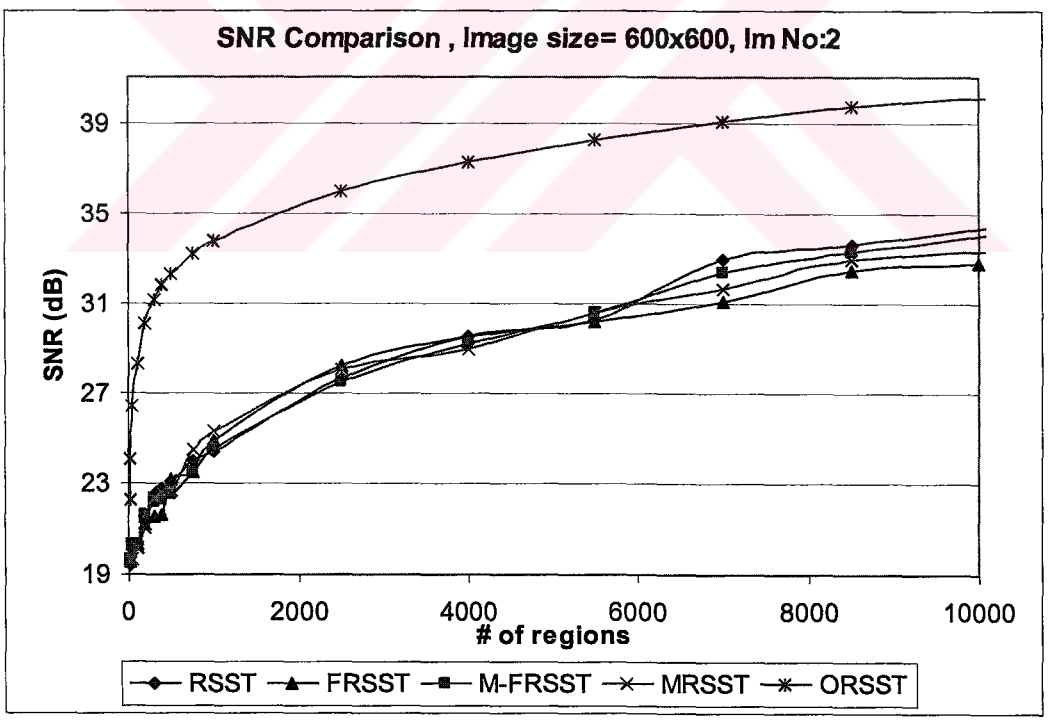


Figure A.18. SNR comparison of image number 2, region number is up to 10,000, image size is 600x600.

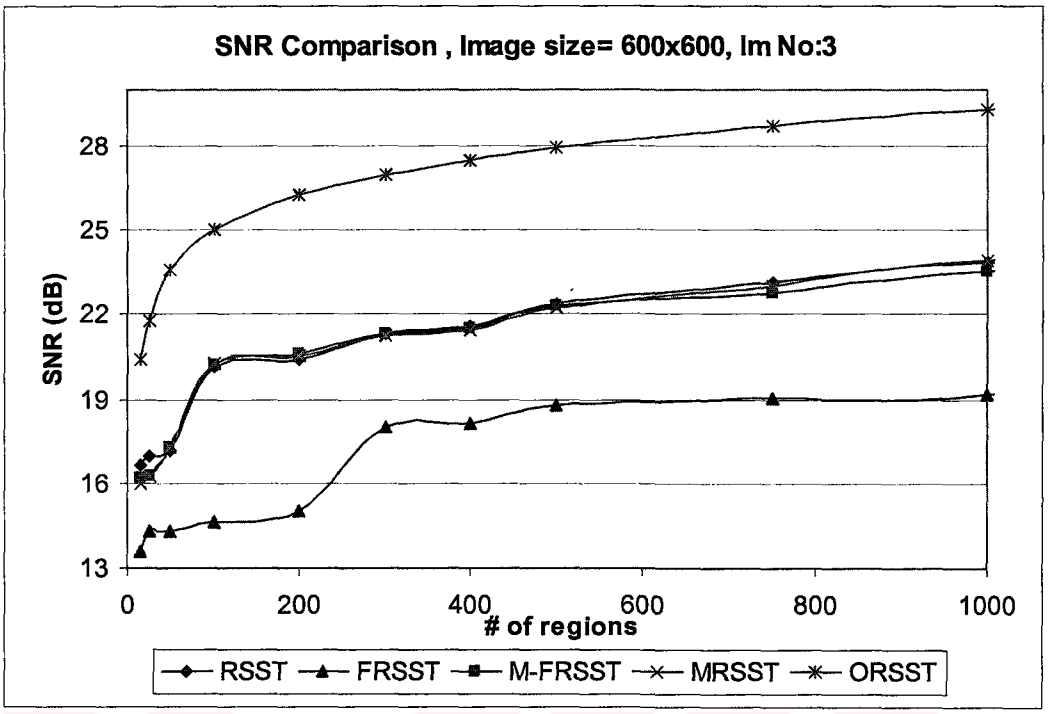


Figure A.19. SNR comparison of image number 3, region number is up to 1000, image size is 600x600.

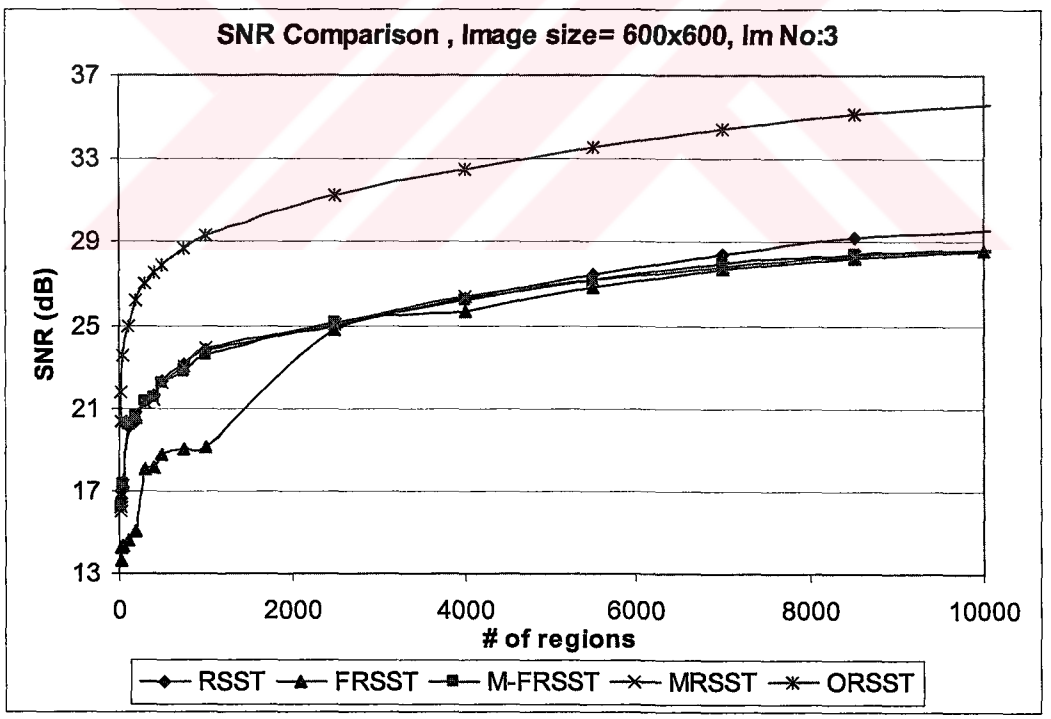


Figure A.20. SNR comparison of image number 3, region number is up to 10,000, image size is 600x600.

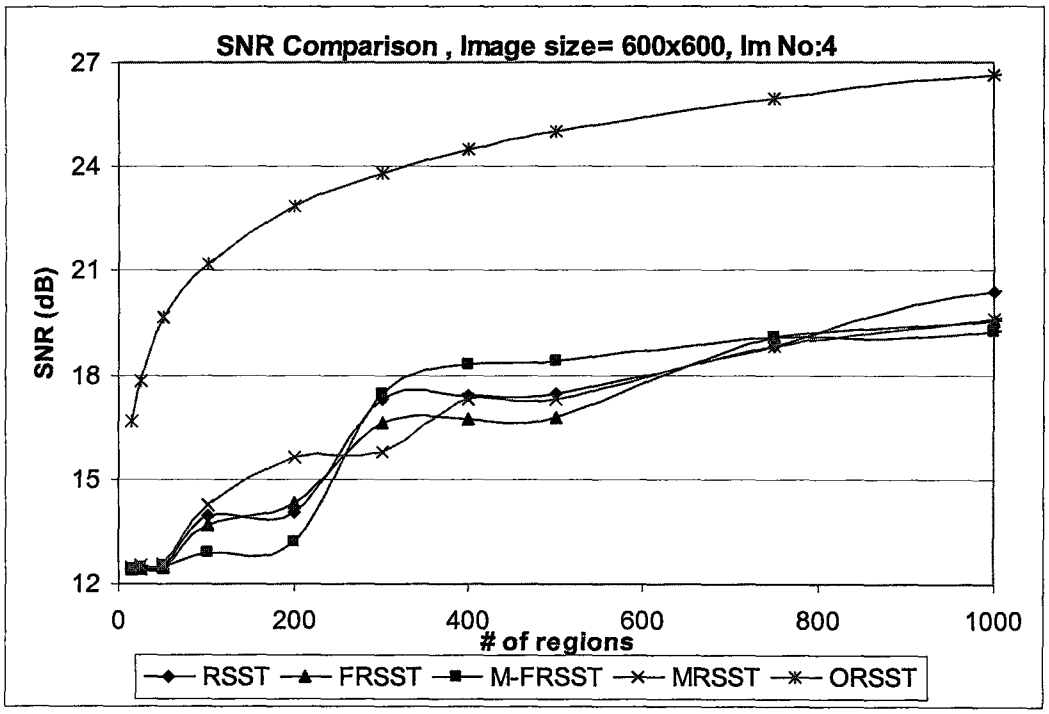


Figure A.21. SNR comparison of image number 4, region number is up to 1000, image size is 600x600.

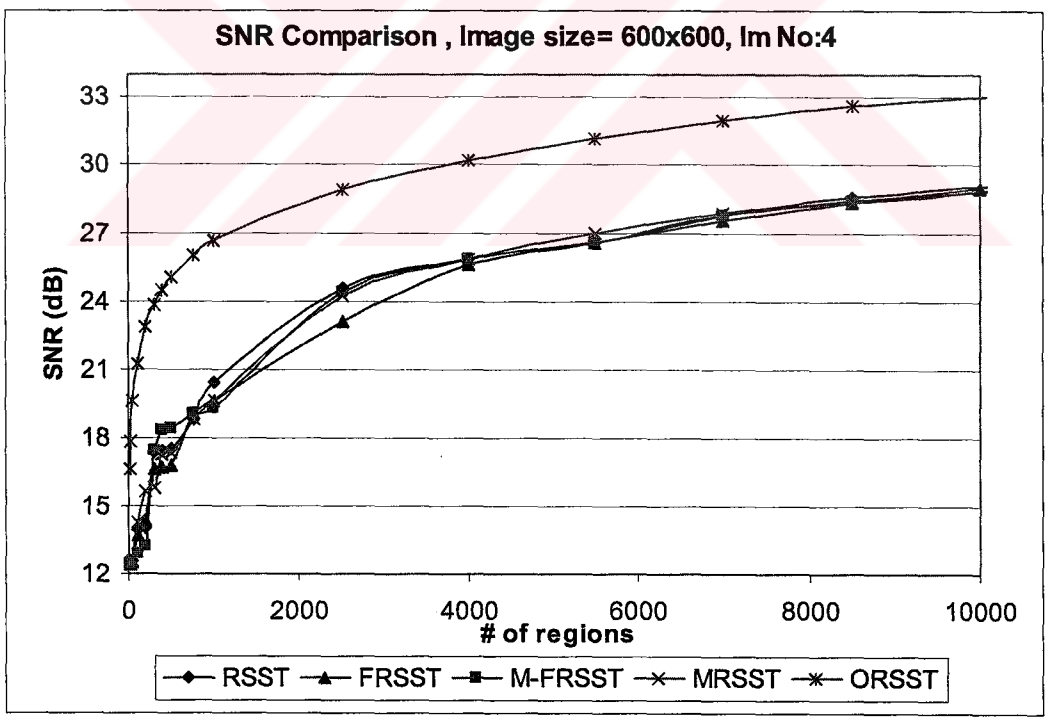


Figure A.22. SNR comparison of image number 4, region number is up to 10,000, image size is 600x600.

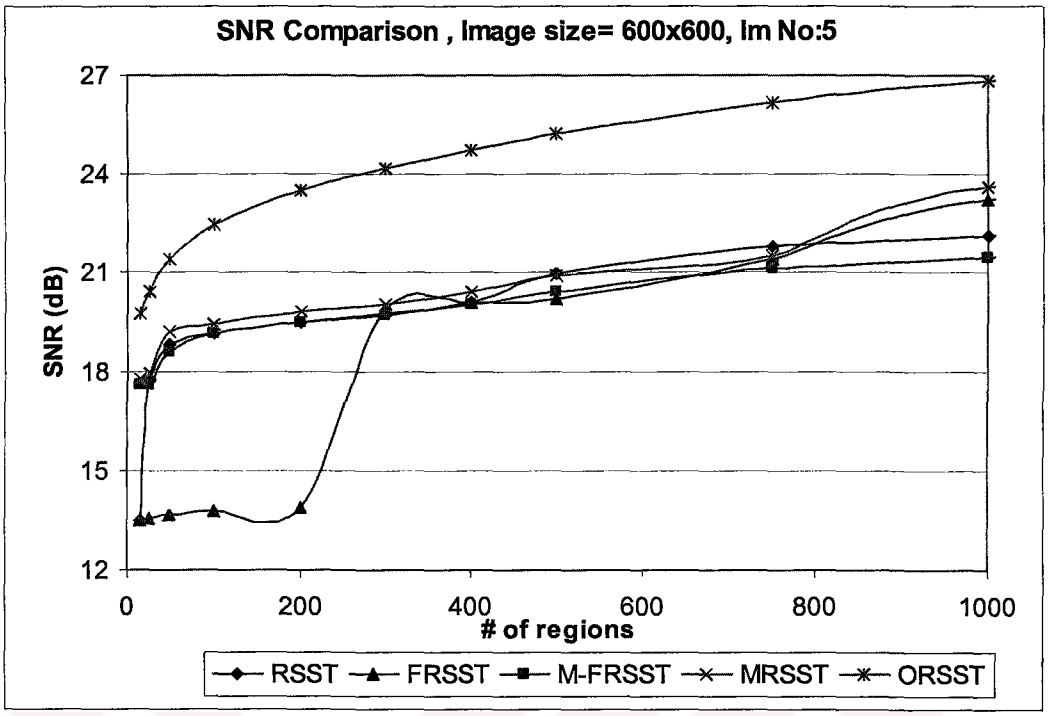


Figure A.23. SNR comparison of image number 5, region number is up to 1000, image size is 600x600.

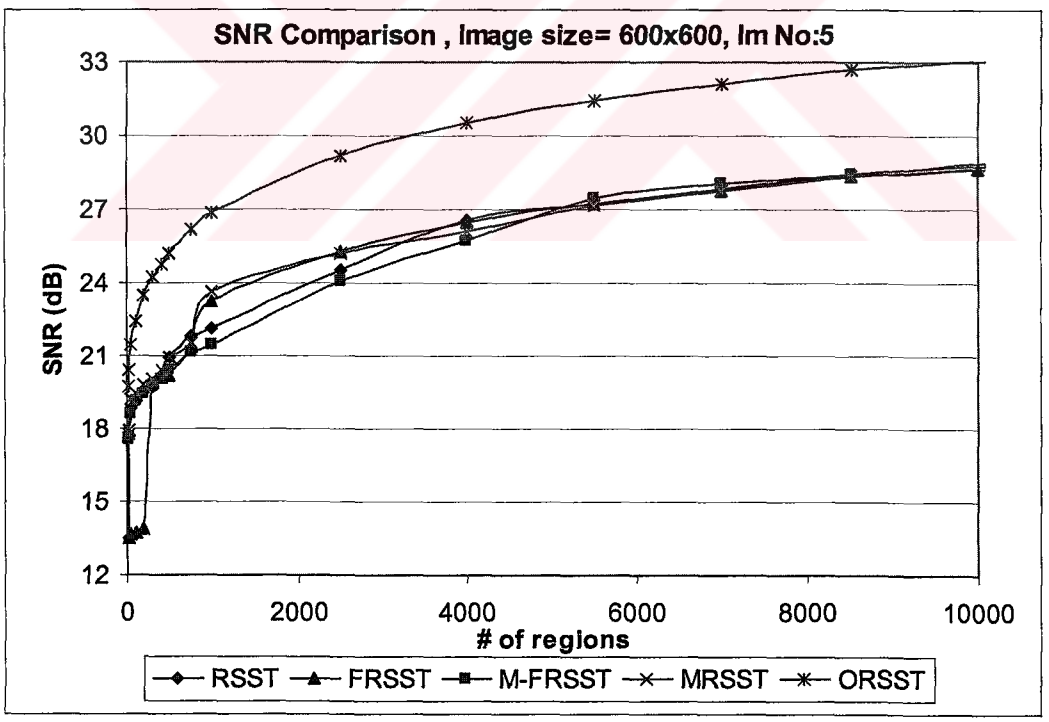


Figure A.24. SNR comparison of image number 5, region number is up to 10,000, image size is 600x600.

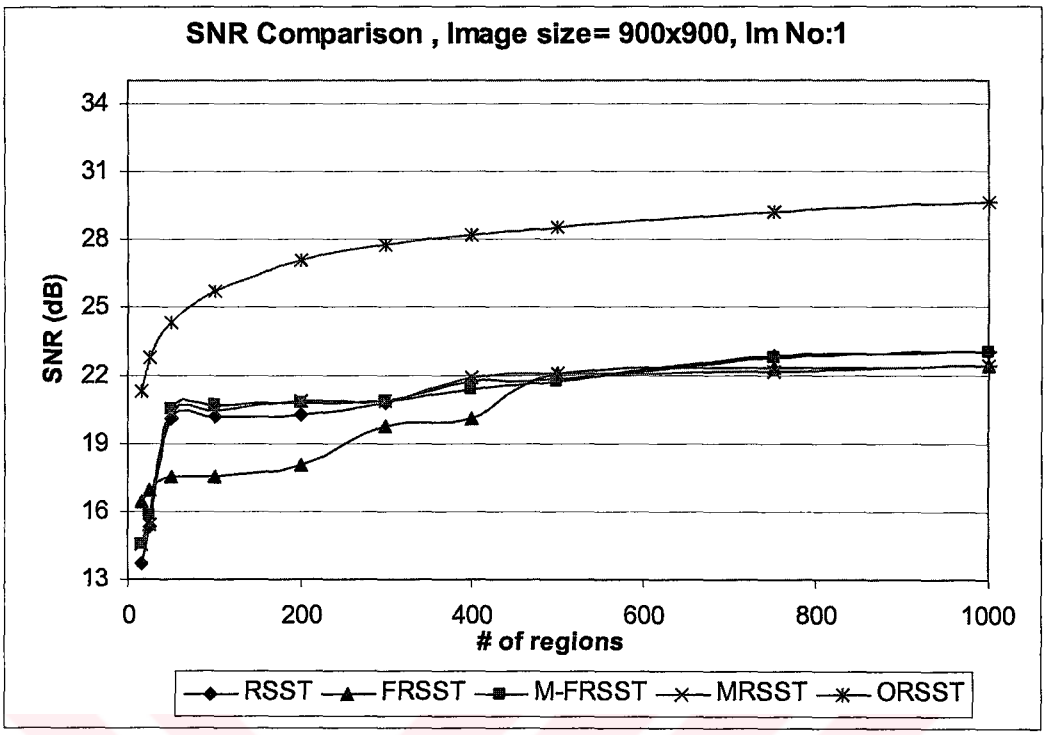


Figure A.25. SNR comparison of image number 1, region number is up to 1000, image size is 900x900.

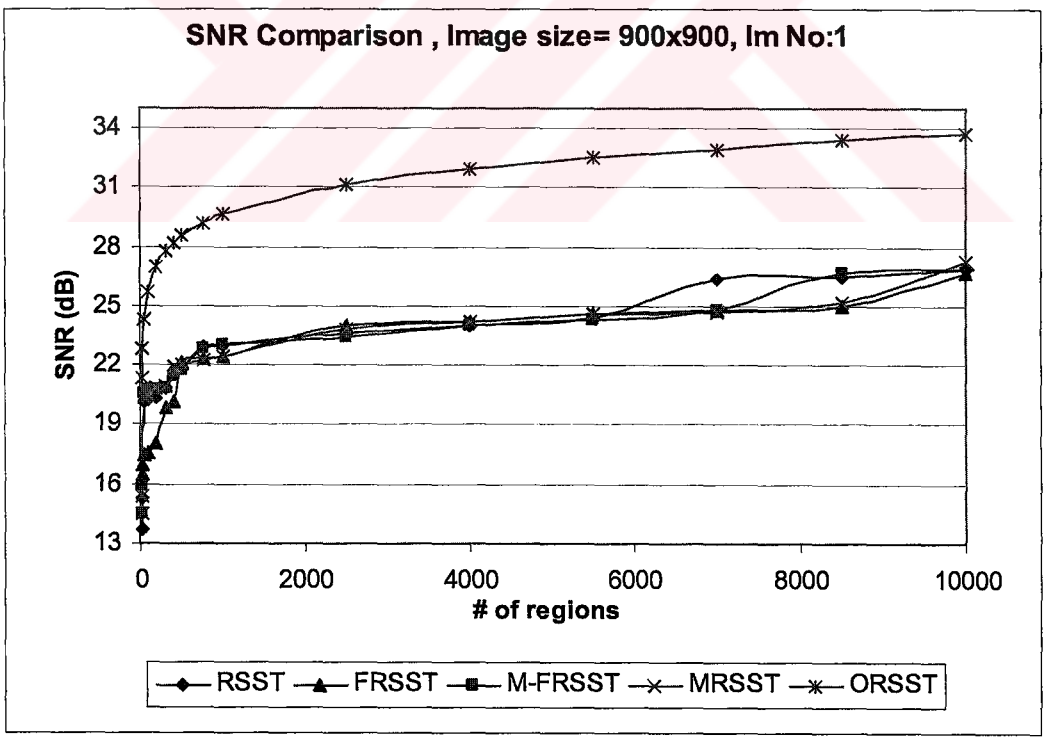


Figure A.26. SNR comparison of image number 1, region number is up to 10,000, image size is 900x900.

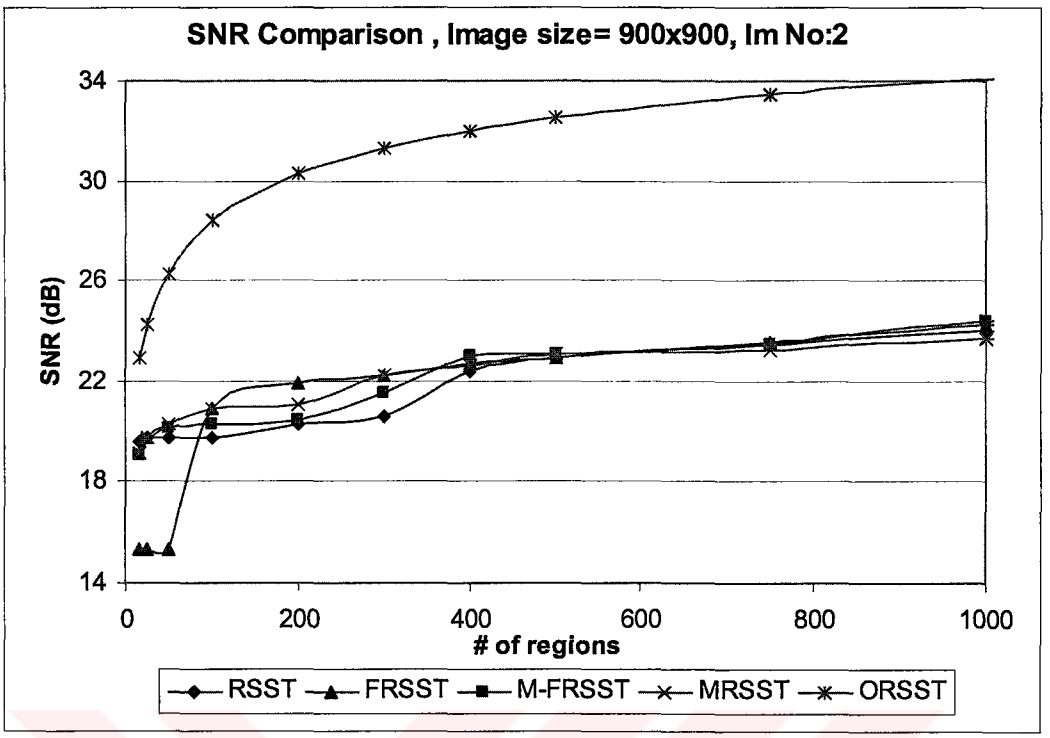


Figure A.27. SNR comparison of image number 2, region number is up to 1000, image size is 900x900.

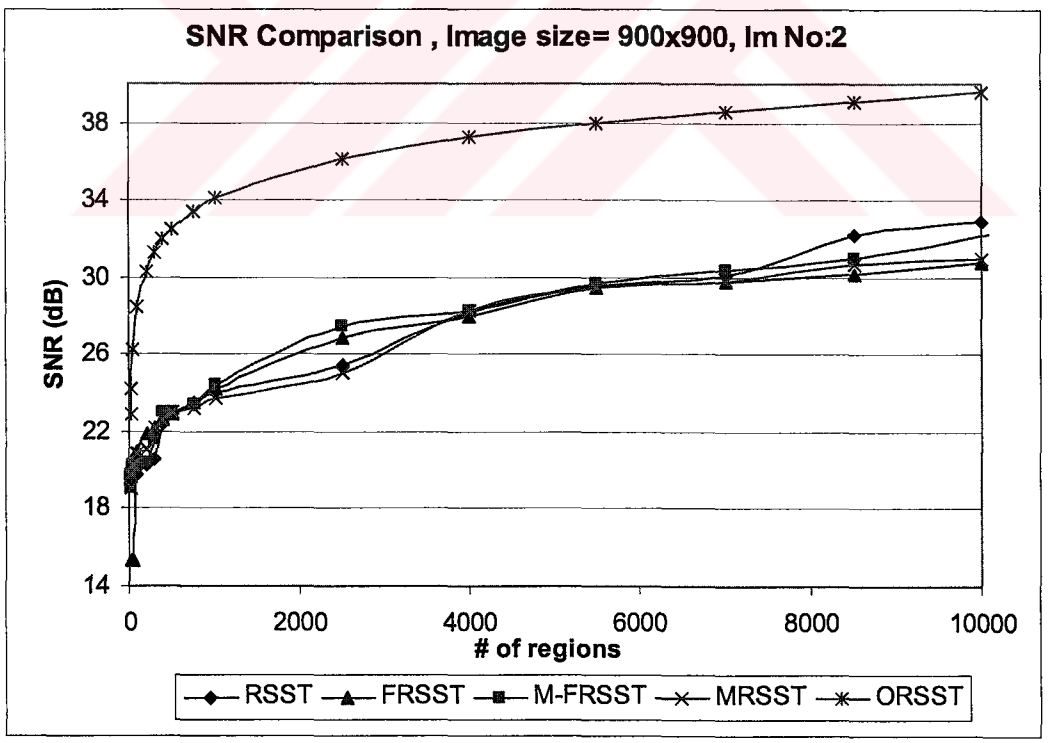


Figure A.28. SNR comparison of image number 2, region number is up to 10,000, image size is 900x900.

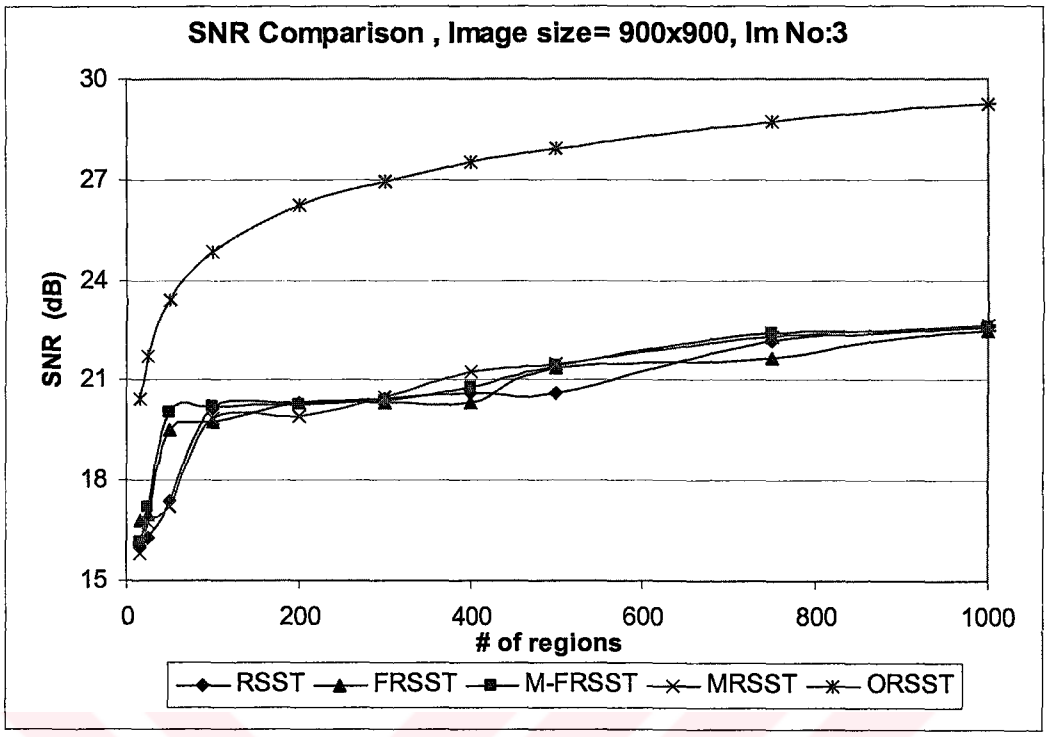


Figure A.29. SNR comparison of image number 3, region number is up to 1000, image size is 900x900.

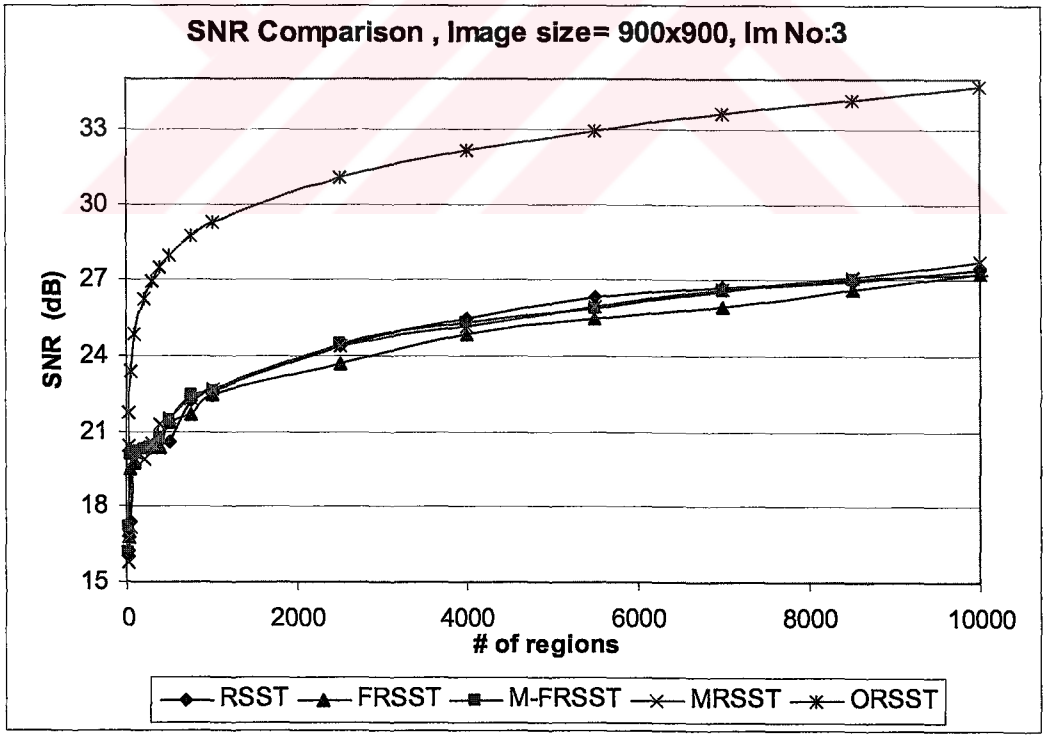


Figure A.30. SNR comparison of image number 3, region number is up to 10,000, image size is 900x900.

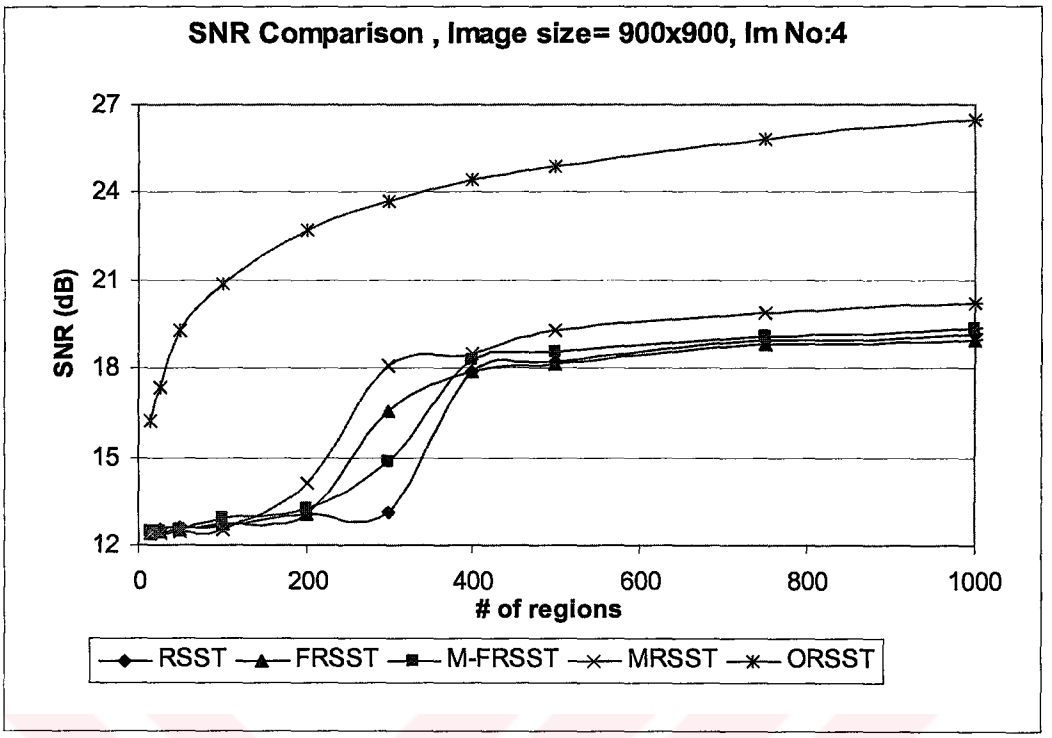


Figure A.31. SNR comparison of image number 4, region number is up to 1000, image size is 900x900.

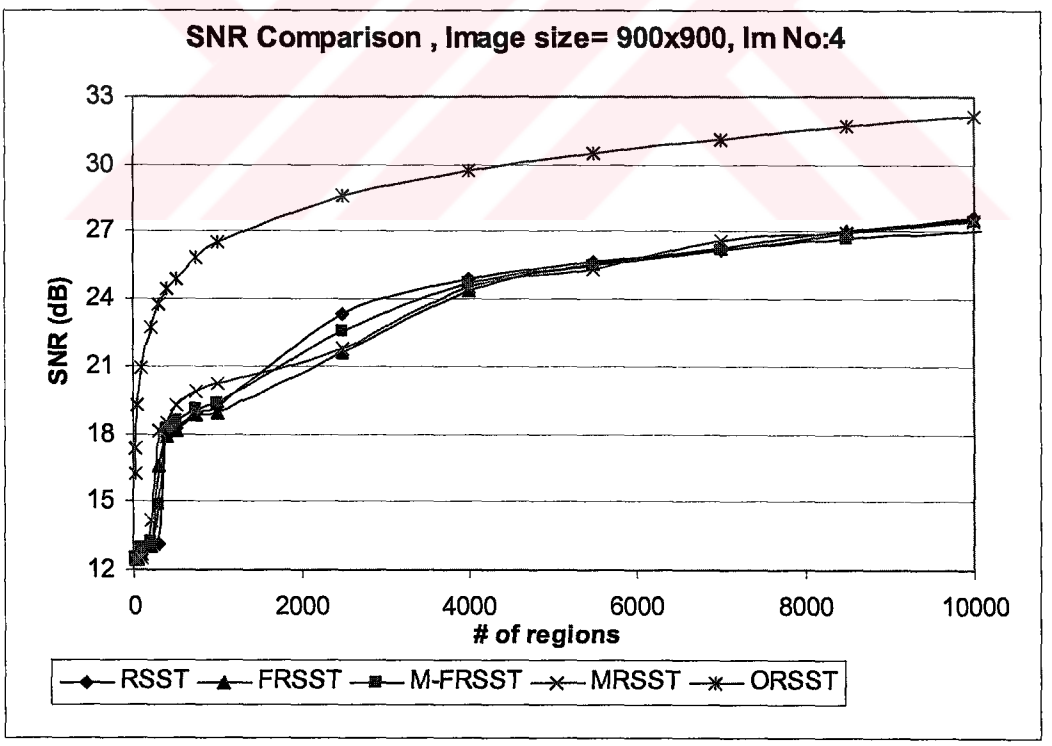


Figure A.32. SNR comparison of image number 4, region number is up to 10,000, image size is 900x900.

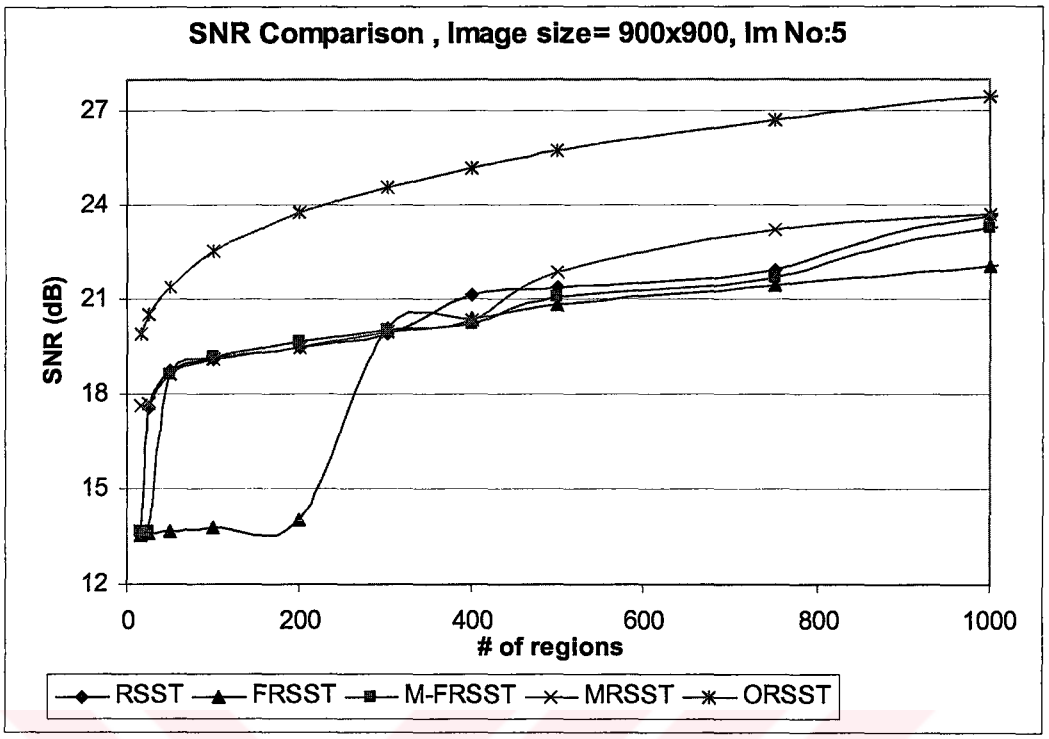


Figure A.33. SNR comparison of image number 5, region number is up to 1000, image size is 900x900.

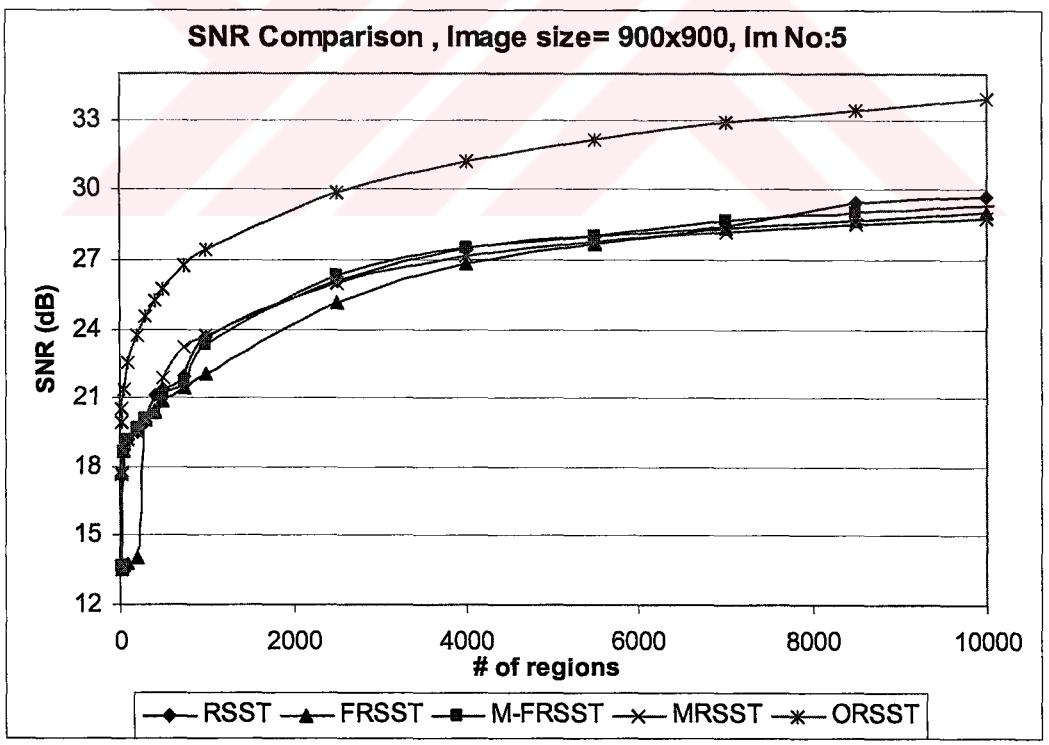


Figure A.34. SNR comparison of image number 5, region number is up to 10,000, image size is 900x900.

E.3. EXECUTION TIME COMPARISON OF DISTRIBUTED ALGORITHM

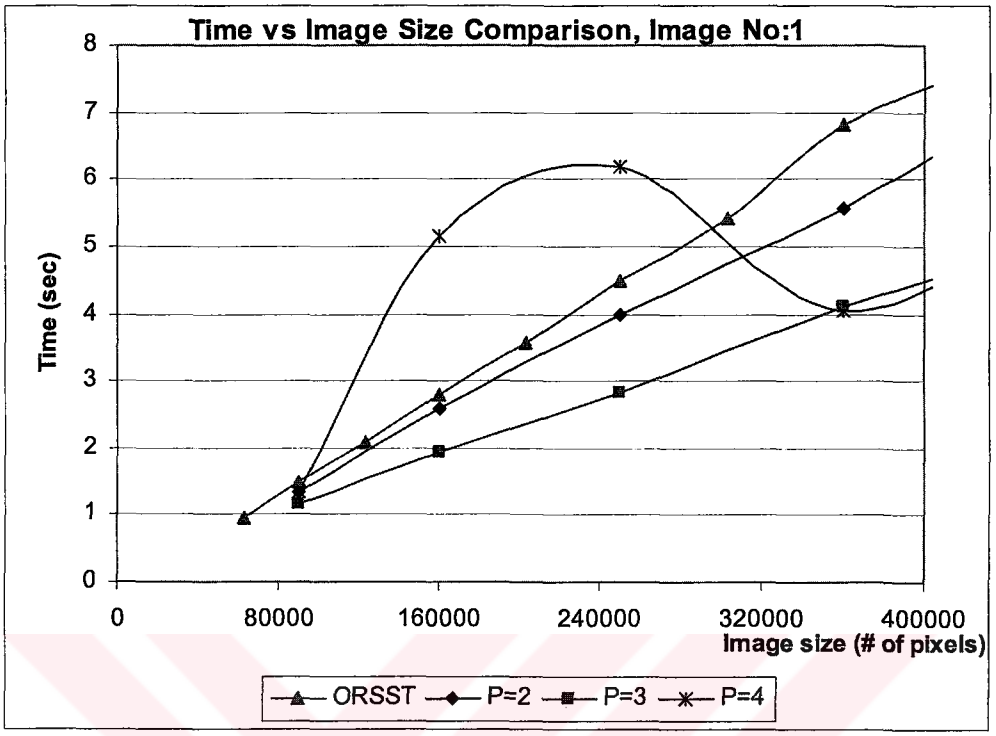


Figure A.35. Execution time comparison of image number 1, image size is up to 400,000 pixels, # of regions=1

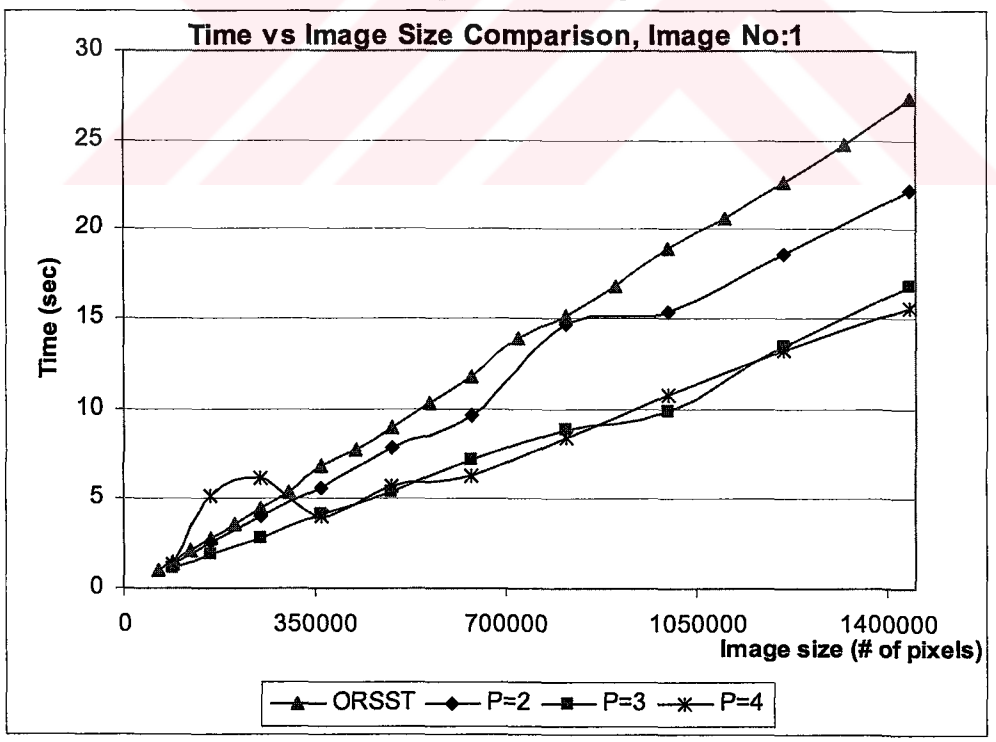


Figure A.36. Execution time comparison of image number 1, image size is up to 1,440,000 pixels, # of regions=1

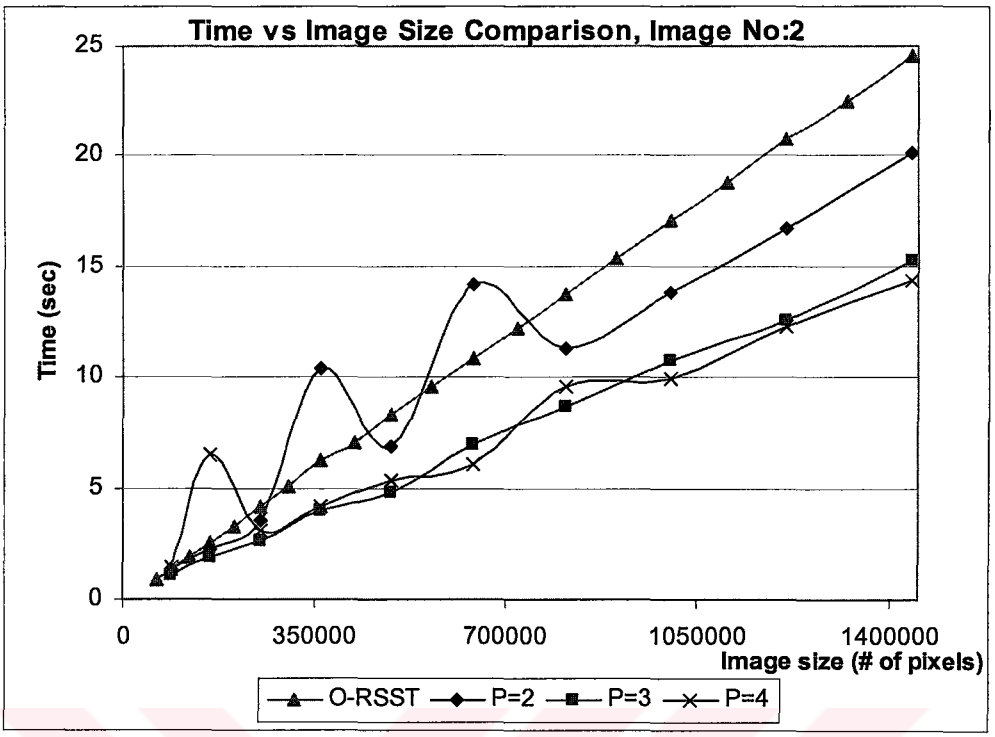


Figure A.37. Execution time comparison of image number 2, image size is up to 400,000 pixels, # of regions=1

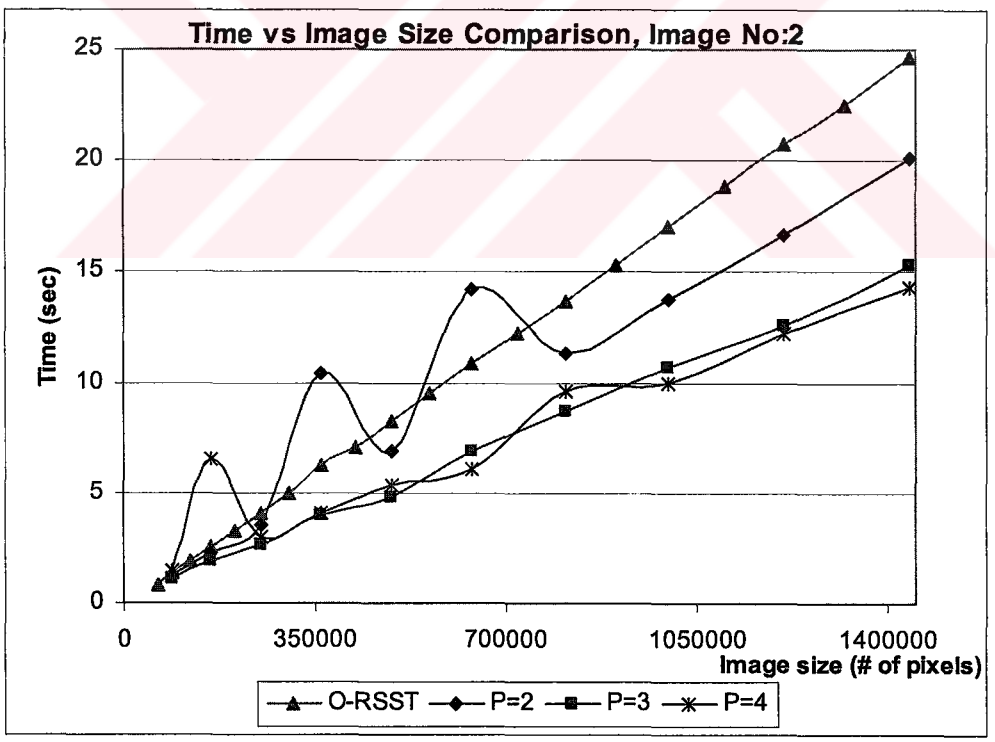


Figure A.38. Execution time comparison of image number 2, image size is up to 1,440,000 pixels, # of regions=1

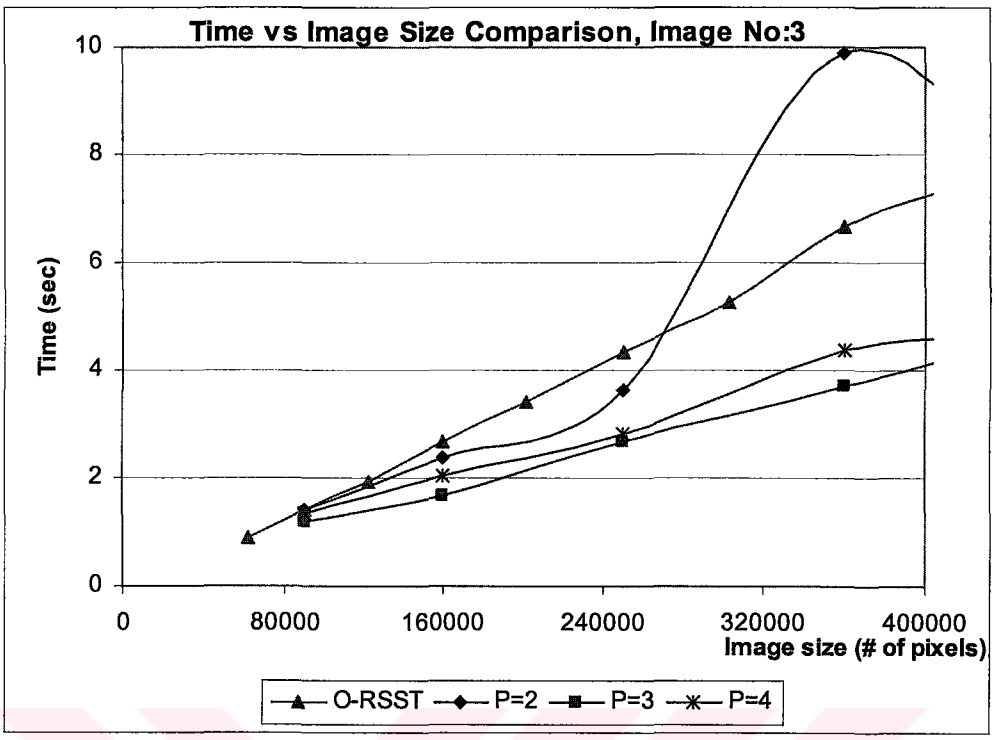


Figure A.39. Execution time comparison of image number 3, image size is up to 400,000 pixels, # of regions=1

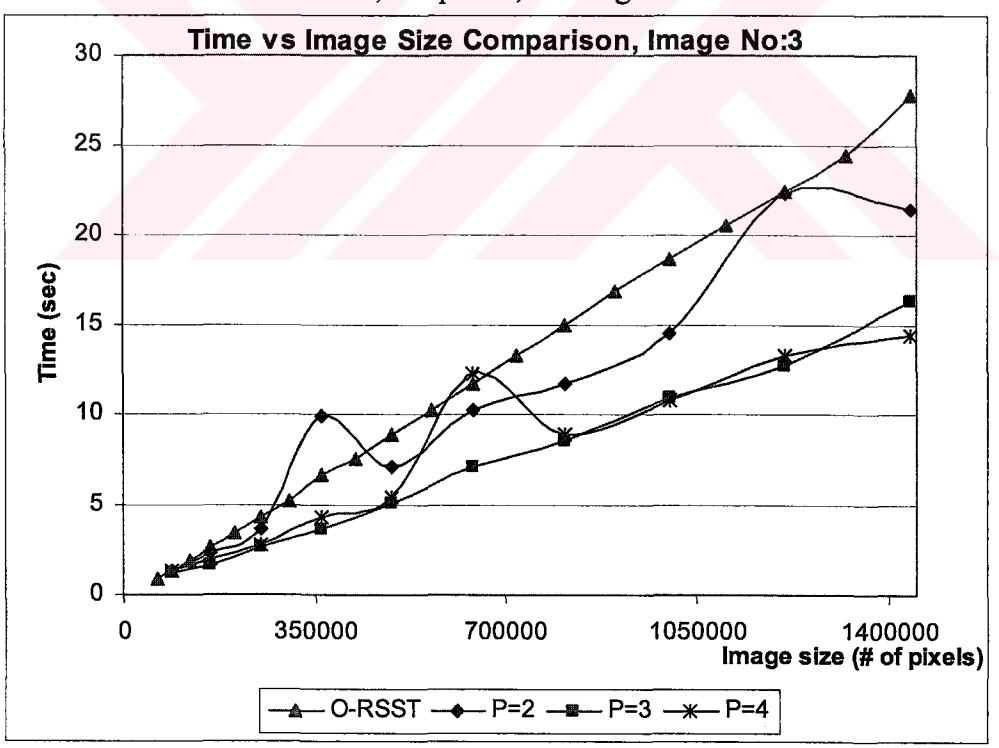


Figure A.40. Execution time comparison of image number 3, image size is up to 1,440,000 pixels, # of regions=1

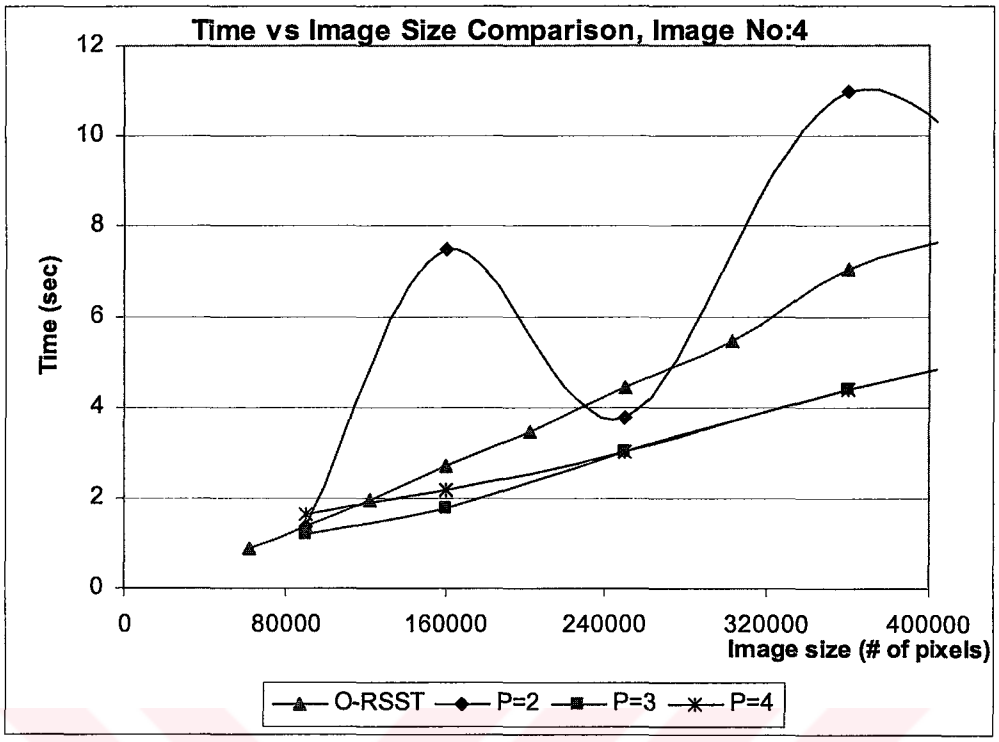


Figure A.41. Execution time comparison of image number 4, image size is up to 400,000 pixels, # of regions=1

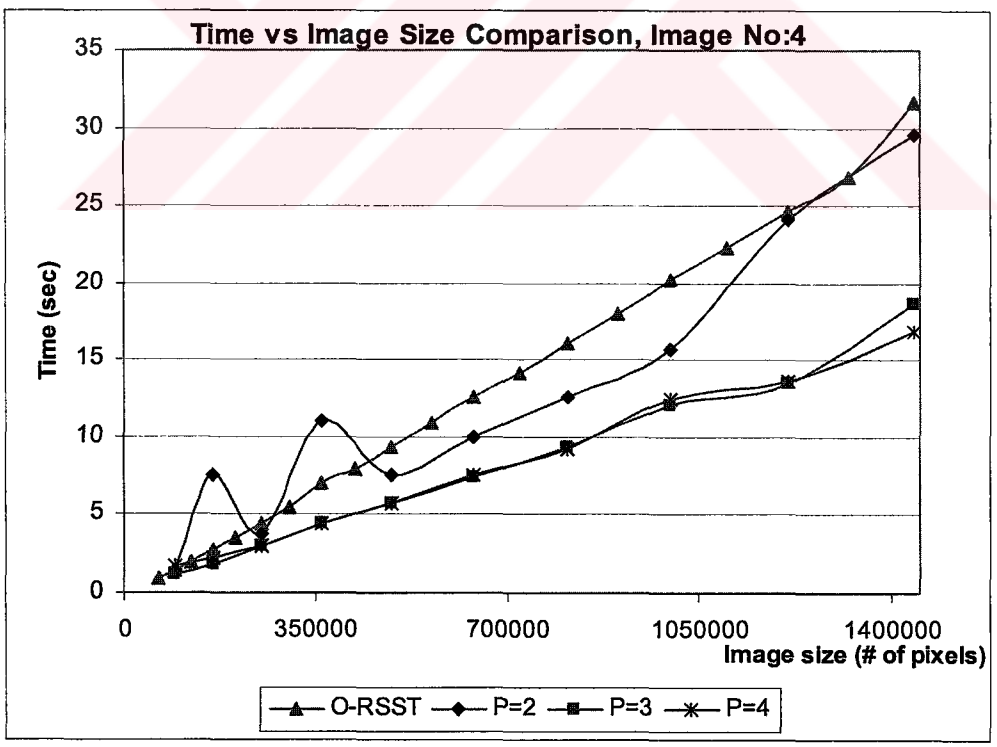


Figure A.42. Execution time comparison of image number 4, image size is up to 1,440,000 pixels, # of regions=1

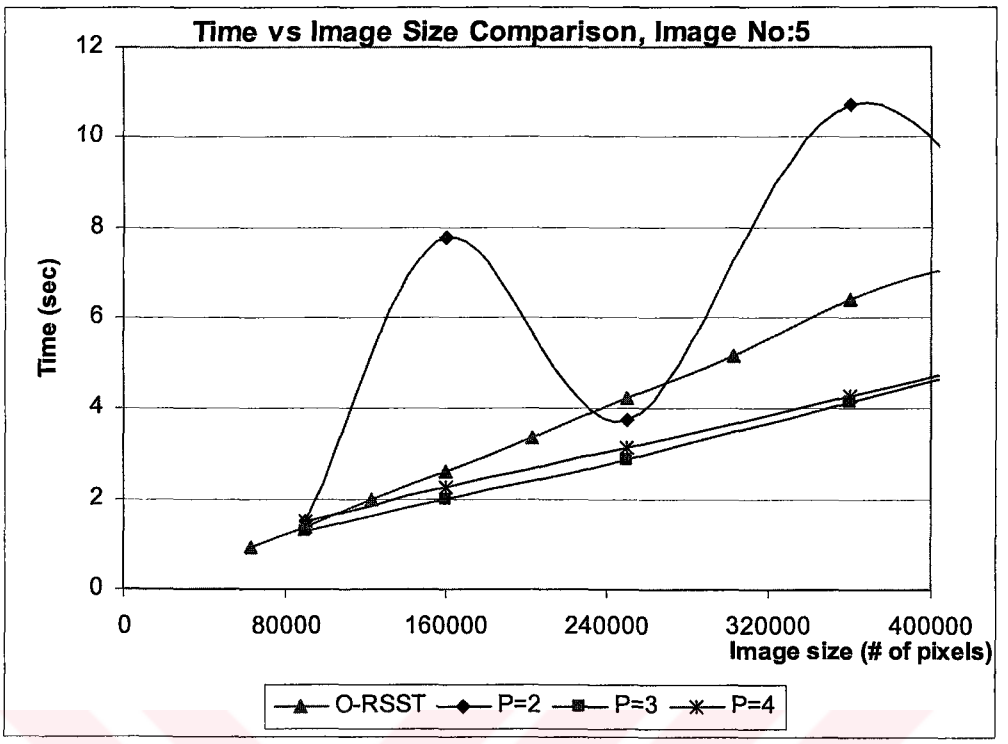


Figure A.43. Execution time comparison of image number 5, image size is up to 400,000 pixels, # of regions=1

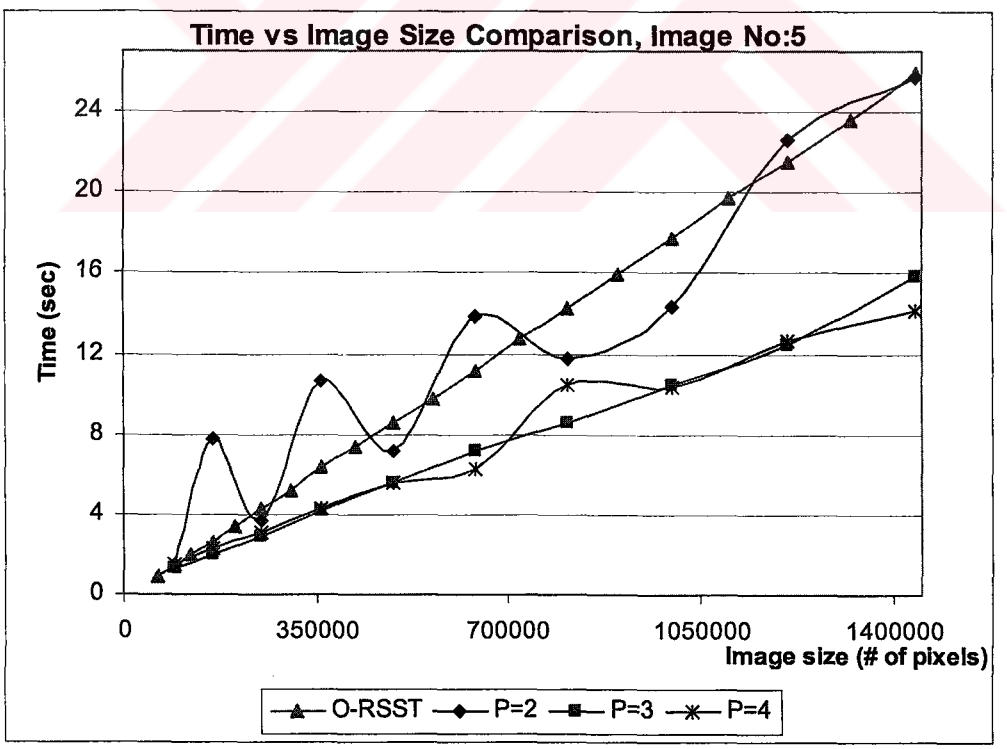


Figure A.44. Execution time comparison of image number 5, image size is up to 1,440,000 pixels, # of regions=1

E.4. SNR COMPARISON OF DISTRIBUTED ALGORITHM

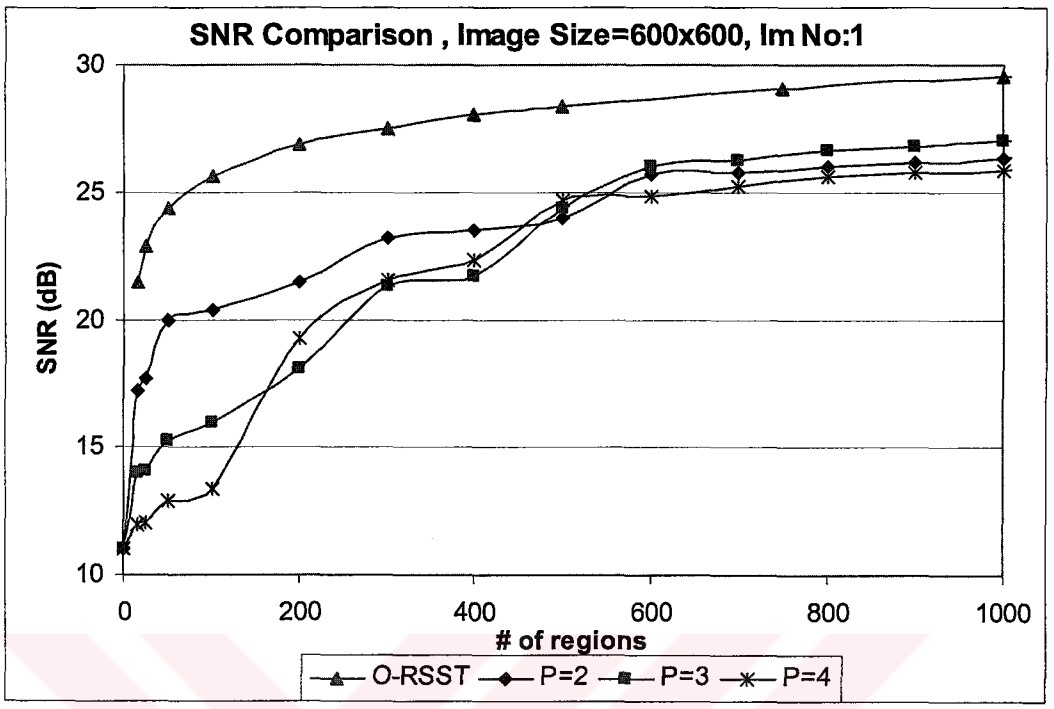


Figure A.45. SNR comparison of image number 1, region number is up to 1000, image size is 600x600.

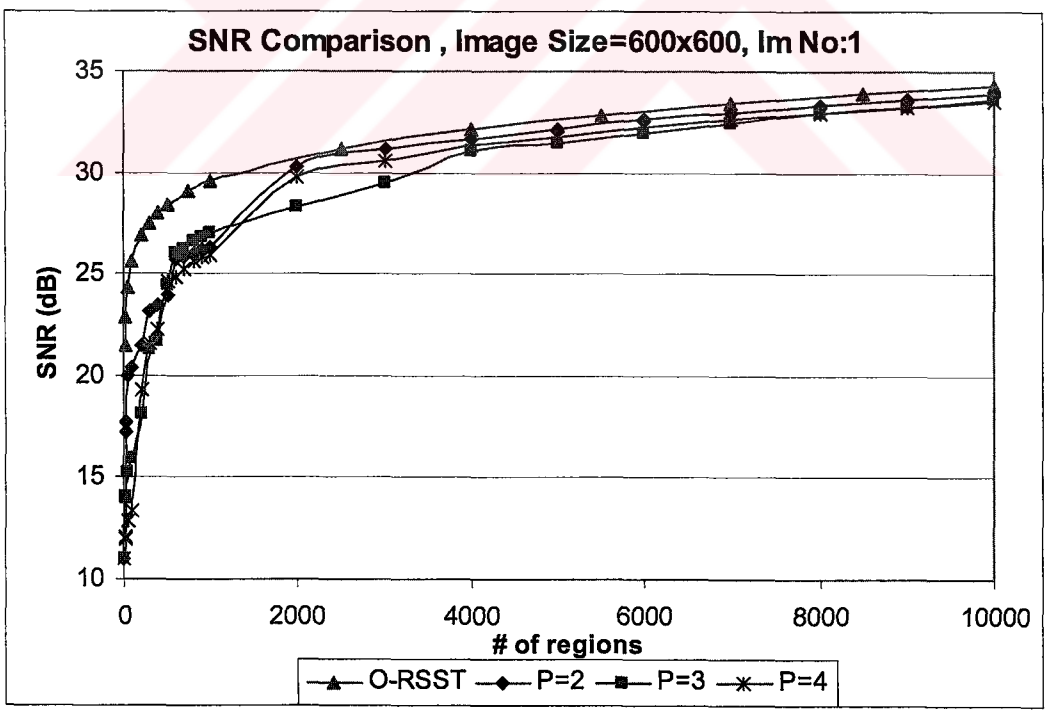


Figure A.46. SNR comparison of image number 1, region number is up to 10,000, image size is 600x600.

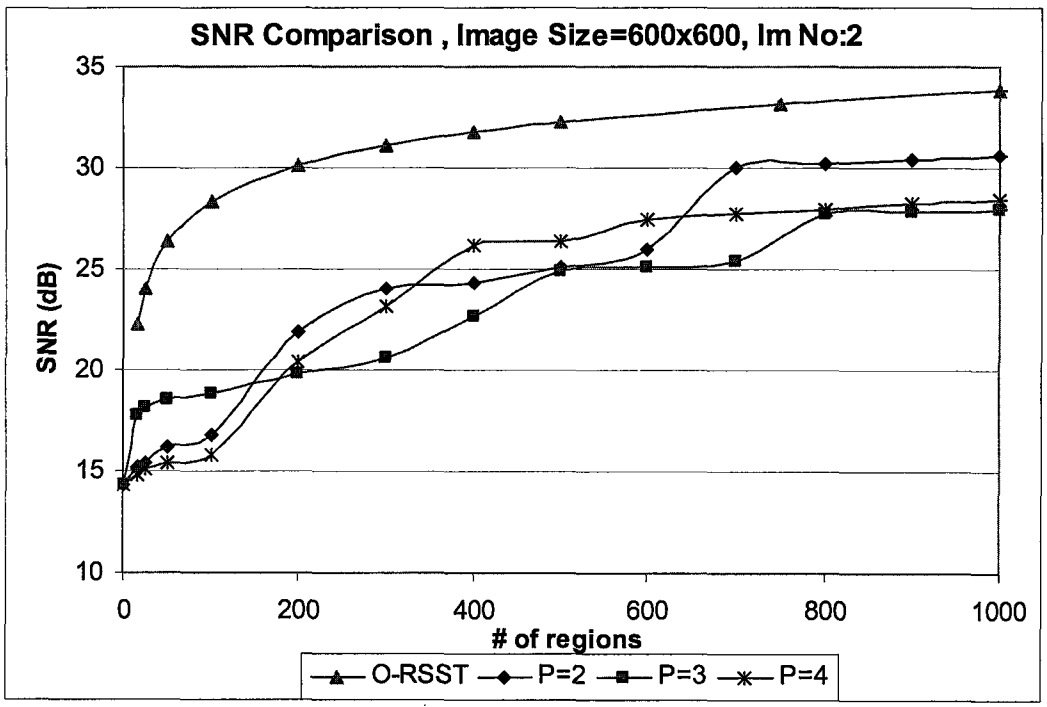


Figure A.47. SNR comparison of image number 2, region number is up to 1000, image size is 600x600.

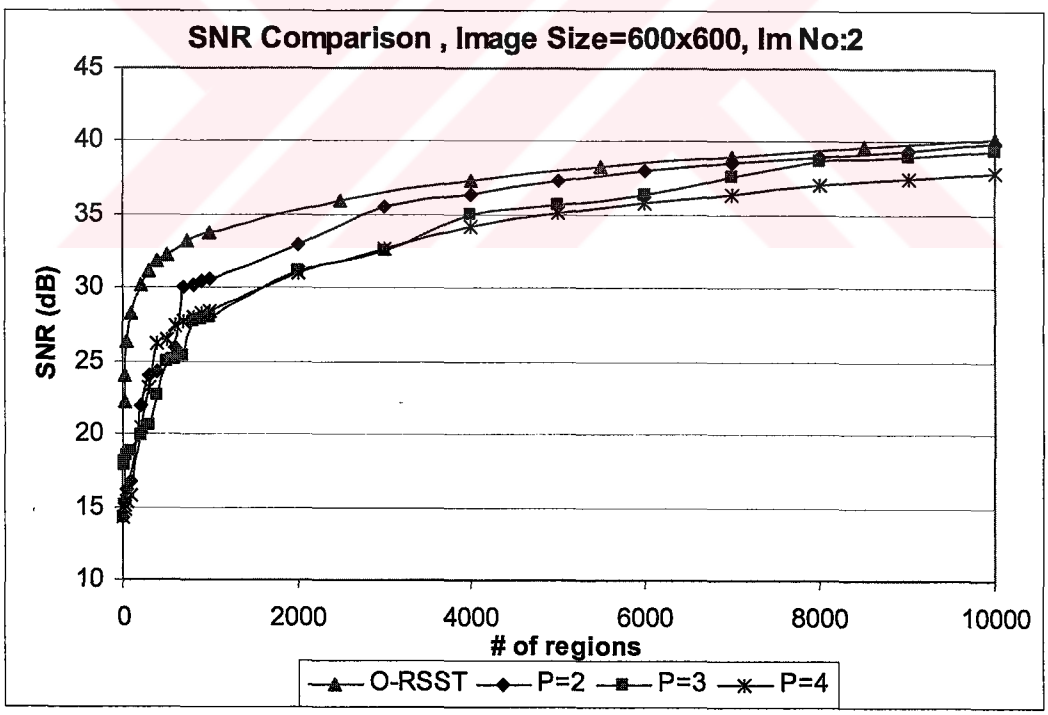


Figure A.48. SNR comparison of image number 2, region number is up to 10,000, image size is 600x600.

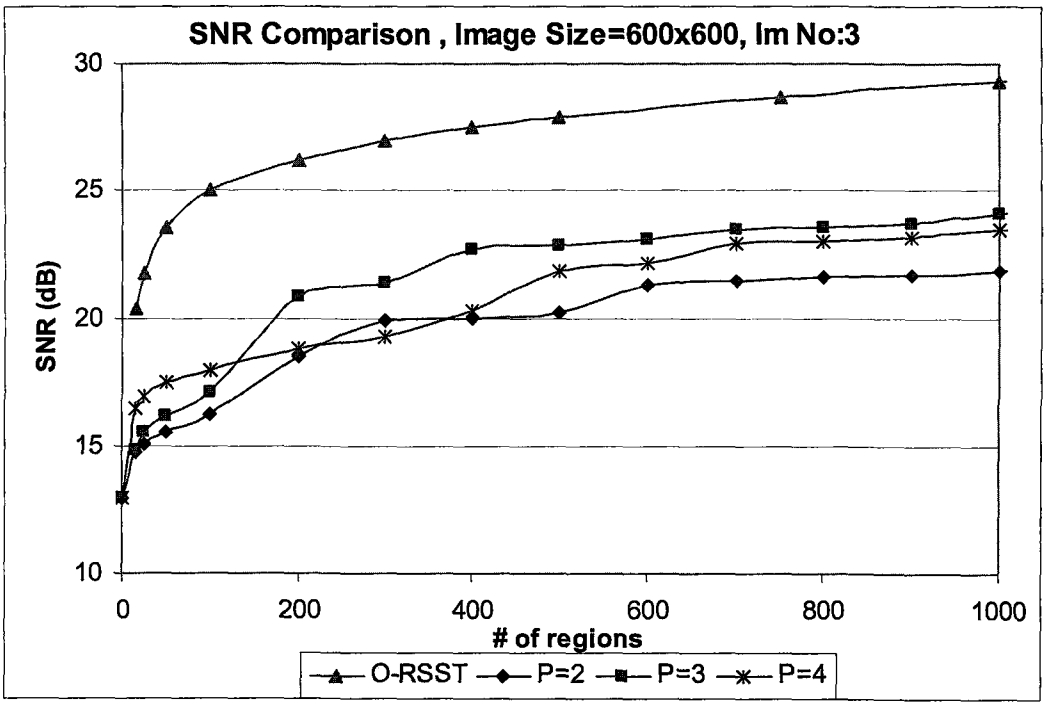


Figure A.49. SNR comparison of image number 3, region number is up to 1000, image size is 600x600.

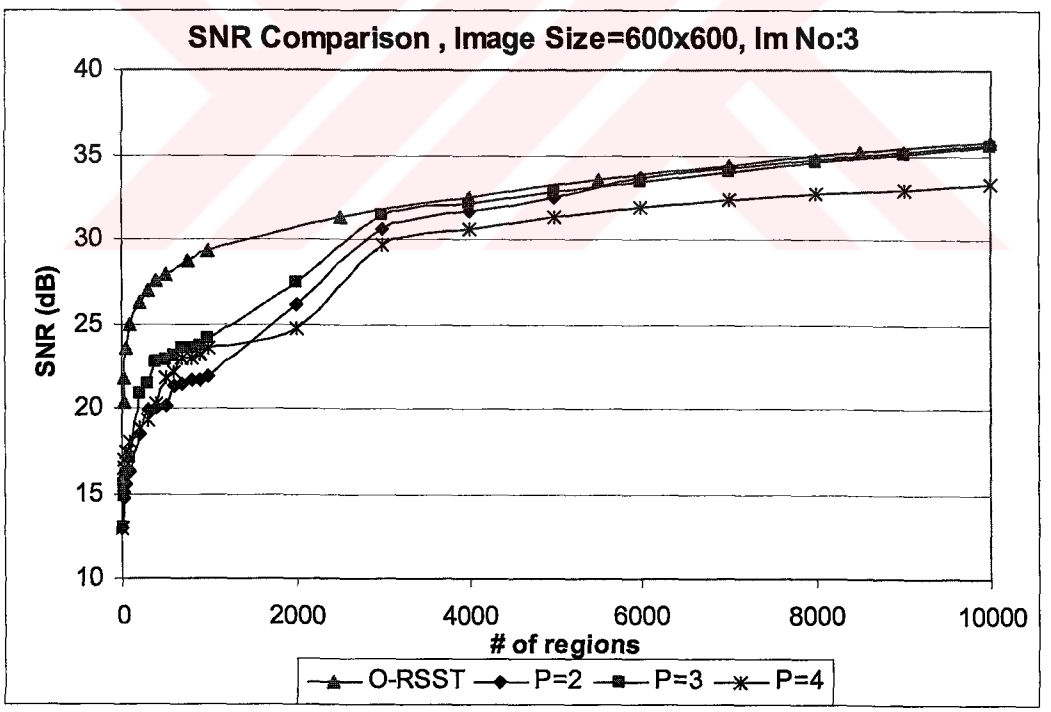


Figure A.50. SNR comparison of image number 3, region number is up to 10,000, image size is 600x600.

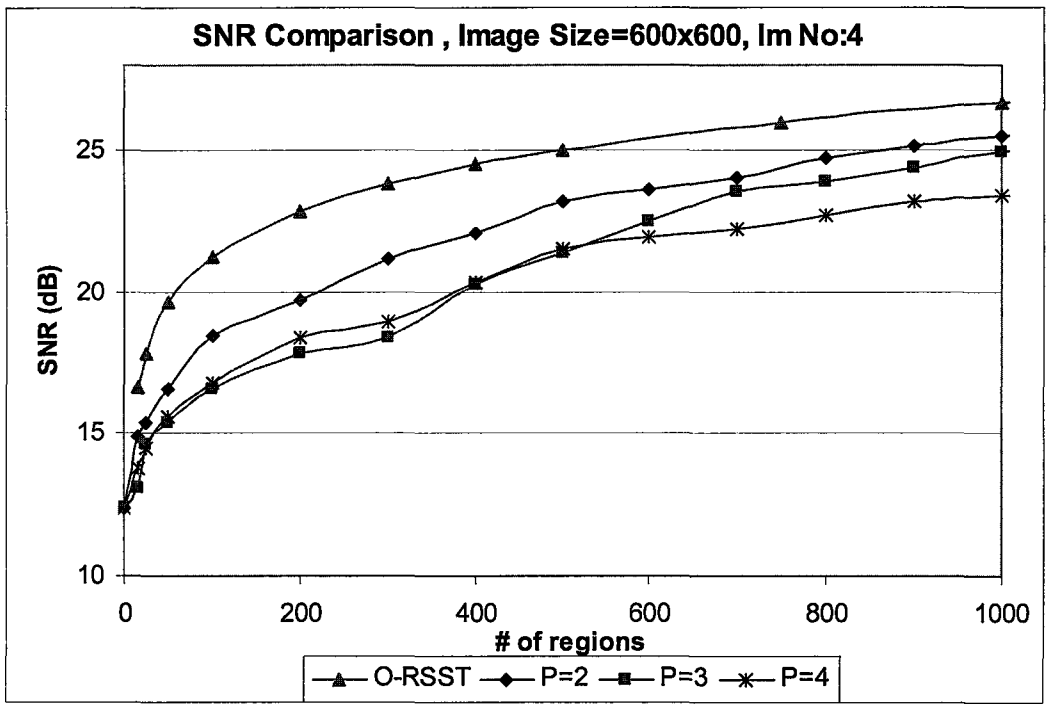


Figure A.51. SNR comparison of image number 4, region number is up to 1000, image size is 600x600.

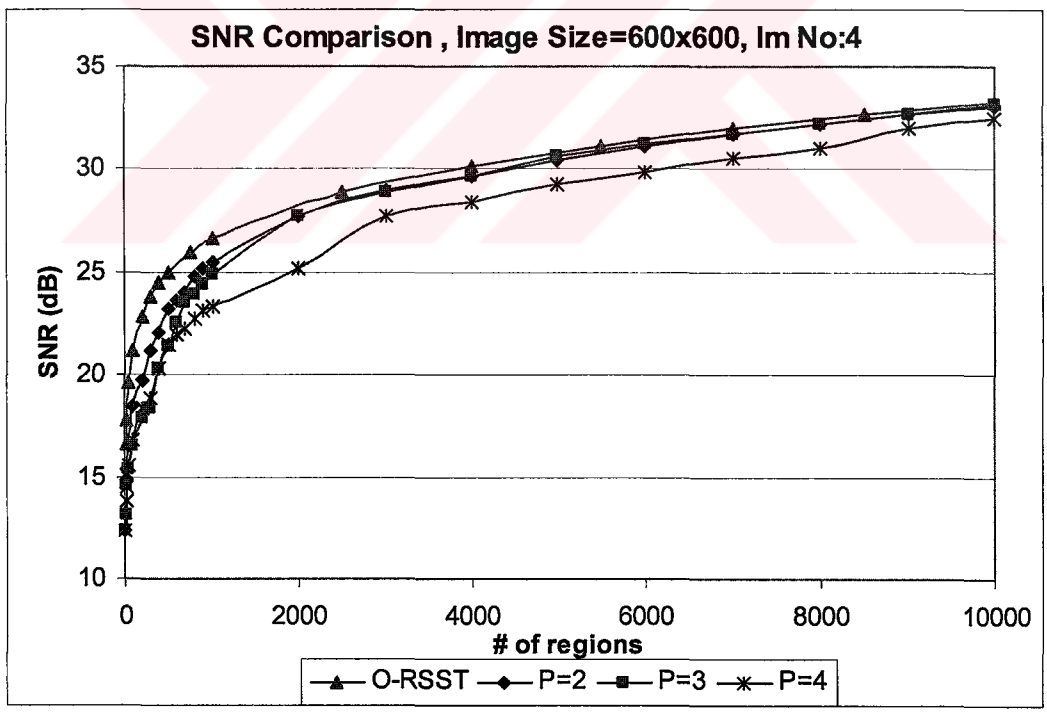


Figure A.52. SNR comparison of image number 4, region number is up to 10,000, image size is 600x600.

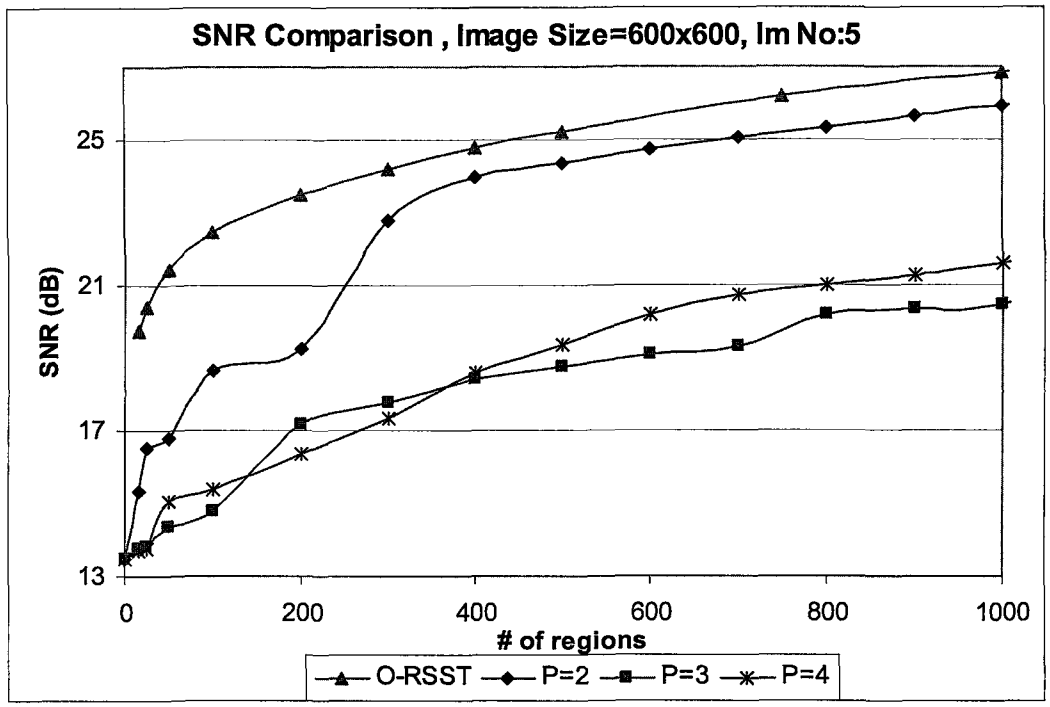


Figure A.53. SNR comparison of image number 5, region number is up to 1000, image size is 600x600.

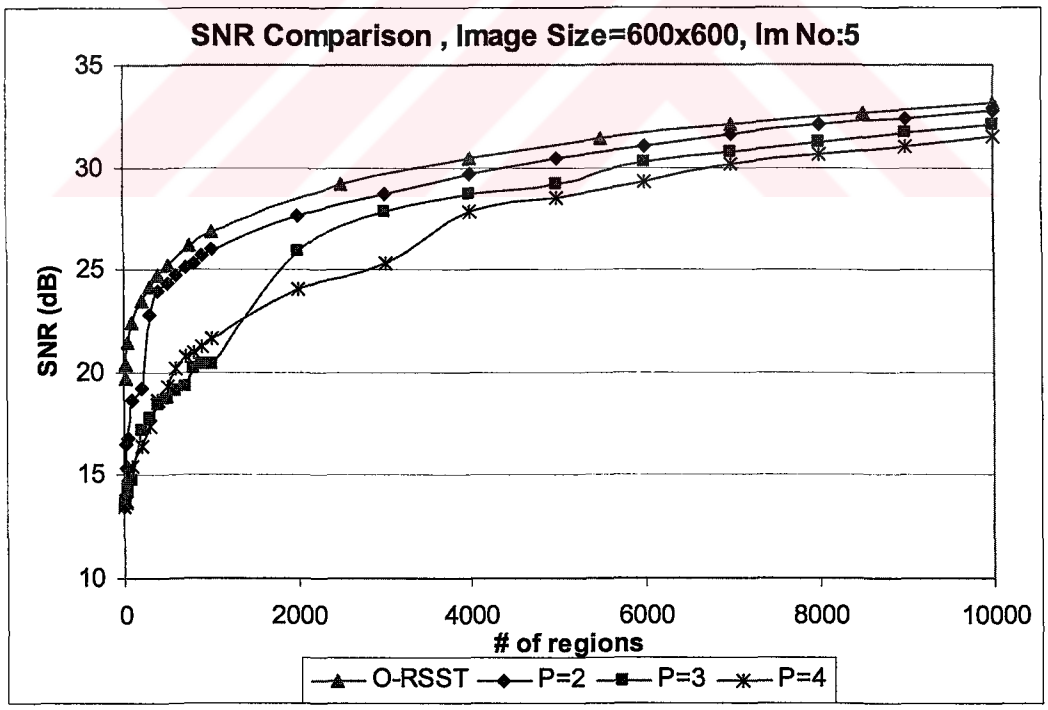


Figure A.54. SNR comparison of image number 5, region number is up to 10,000, image size is 600x600.

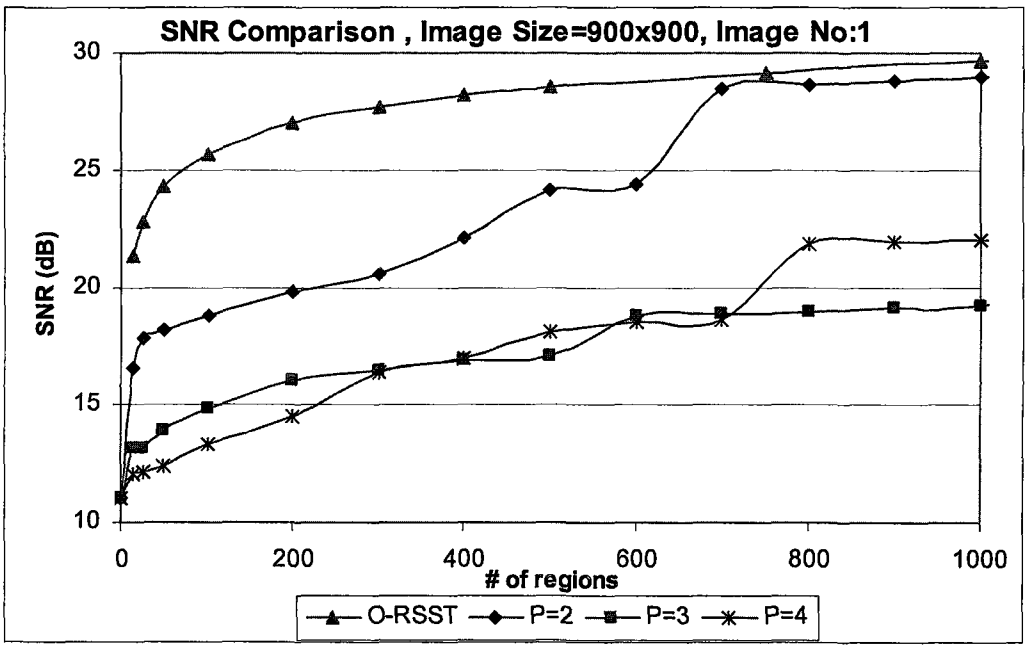


Figure A.55. SNR comparison of image number 1, region number is up to 1000, image size is 900x900.

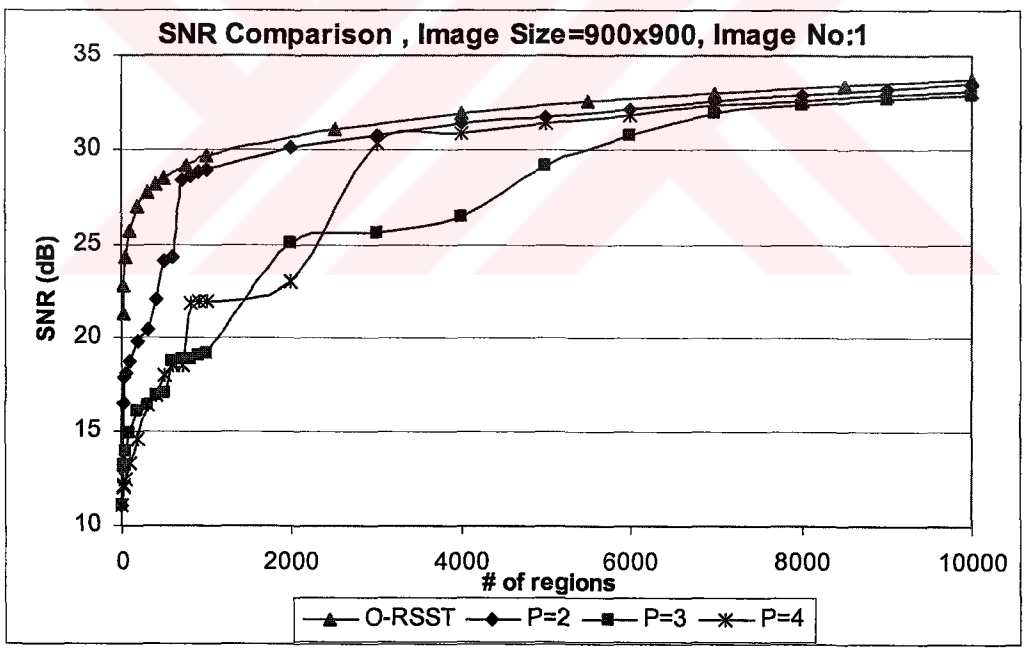


Figure A.56. SNR comparison of image number 1, region number is up to 10,000, image size is 900x900.

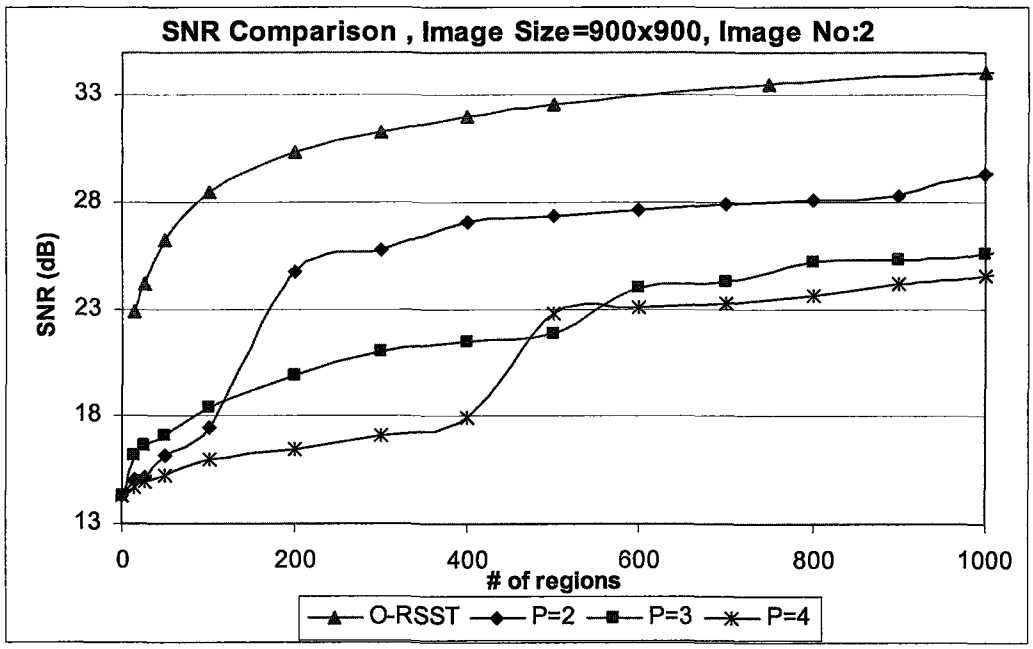


Figure A.57. SNR comparison of image number 2, region number is up to 1000, image size is 900x900.

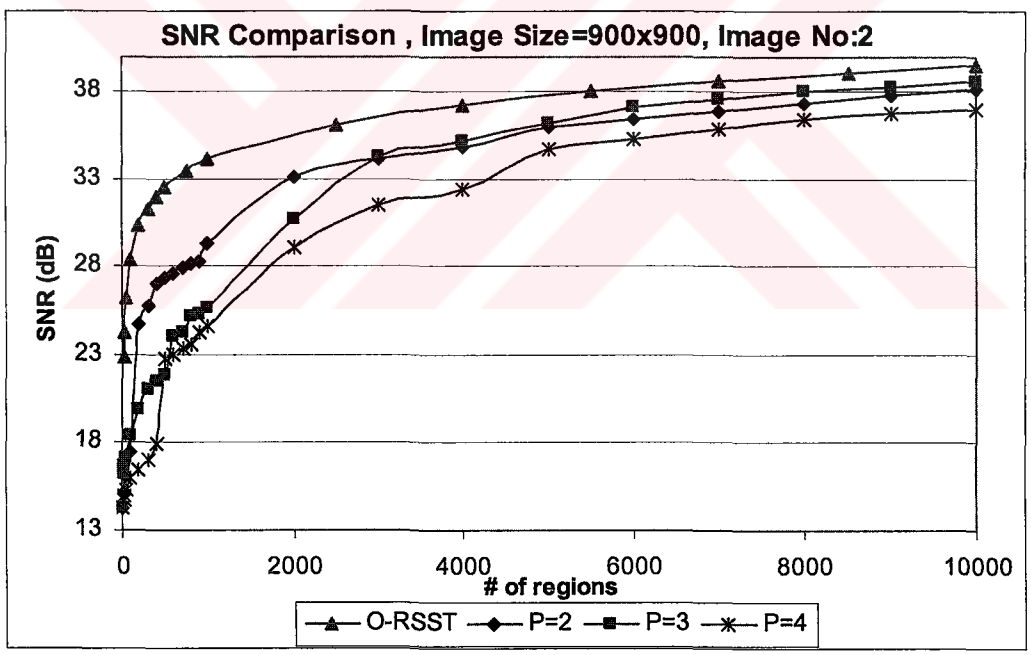


Figure A.58. SNR comparison of image number 2, region number is up to 10,000, image size is 900x900.

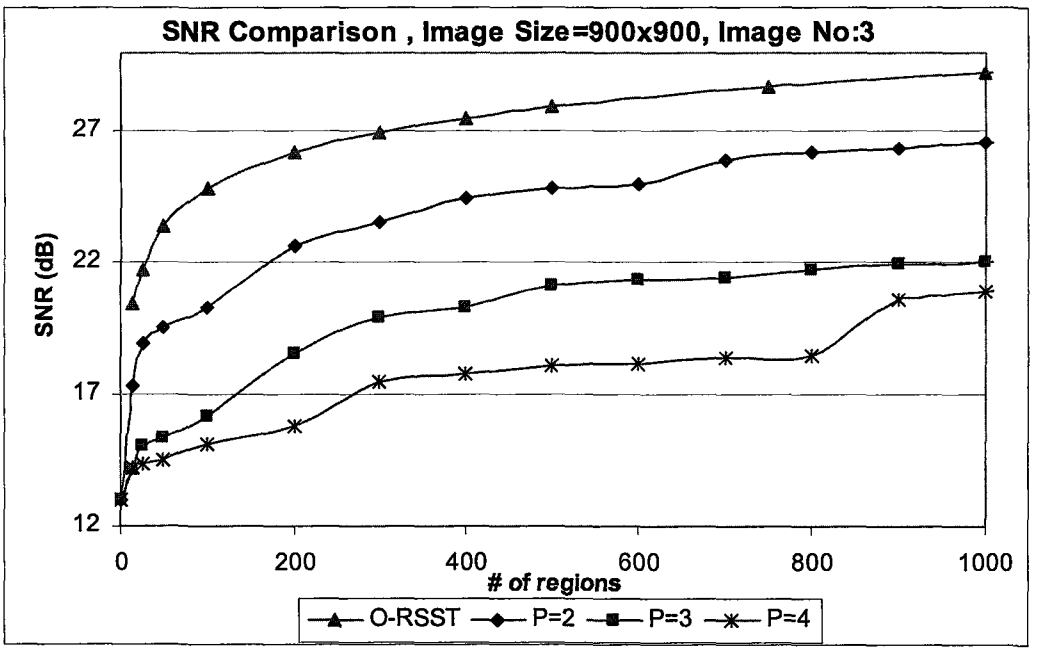


Figure A.59. SNR comparison of image number 3, region number is up to 1000, image size is 900x900.

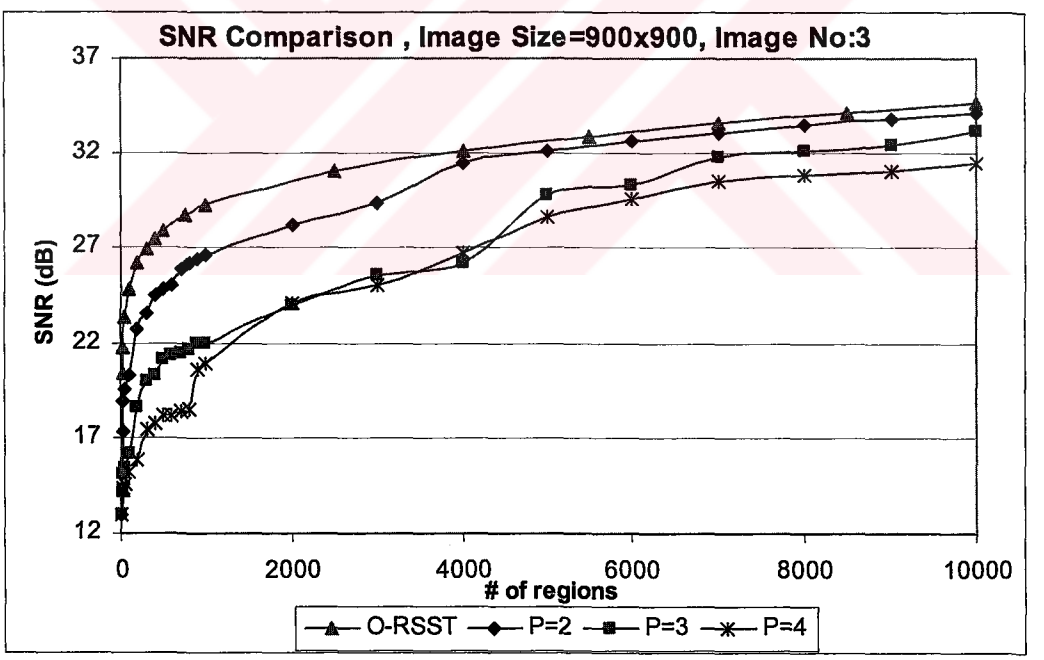


Figure A.60. SNR comparison of image number 3, region number is up to 10,000, image size is 900x900.

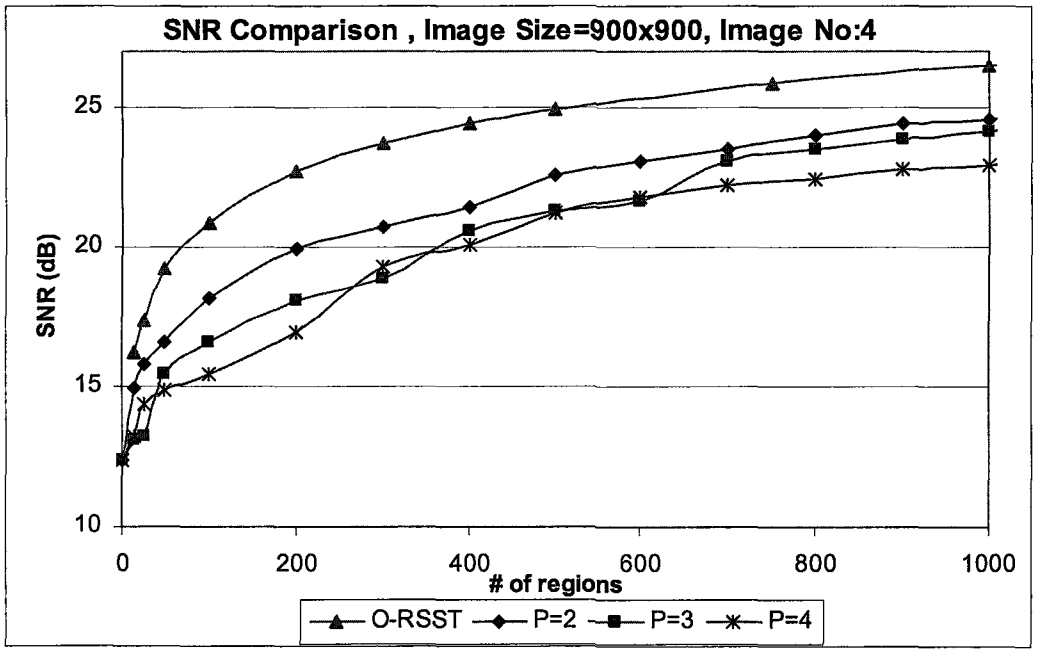


Figure A.61. SNR comparison of image number 4, region number is up to 1000, image size is 900x900.

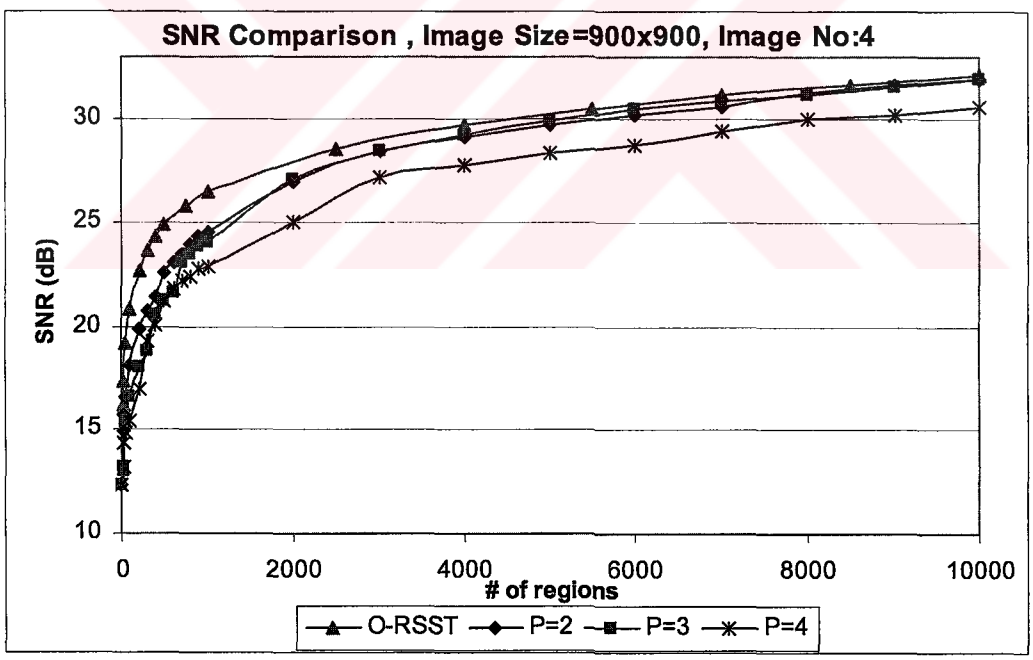


Figure A.62. SNR comparison of image number 4, region number is up to 10,000, image size is 900x900.

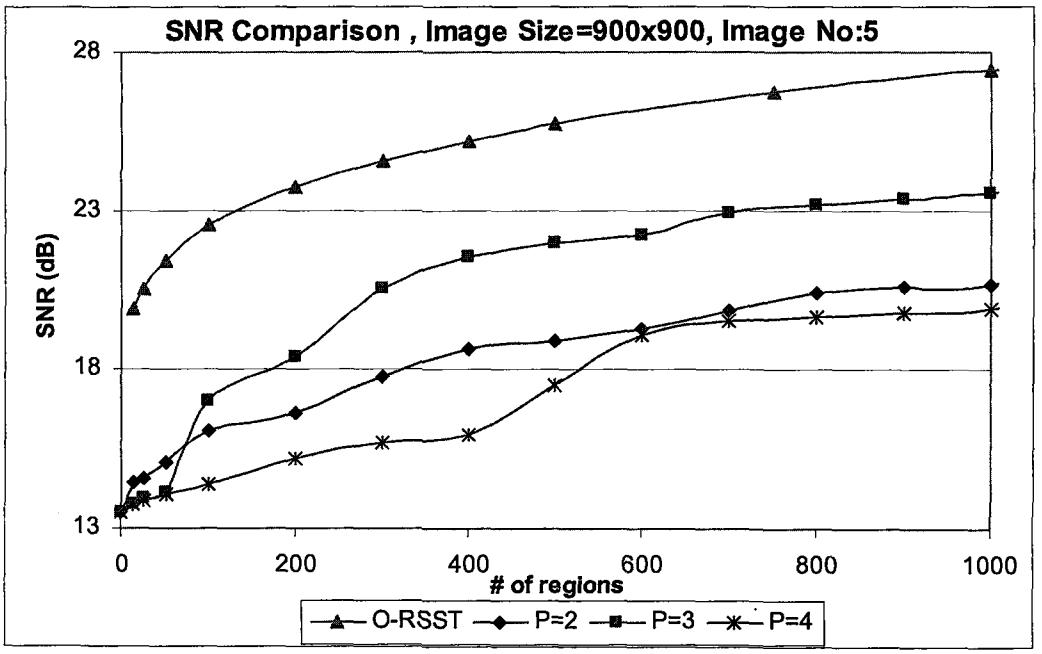


Figure A.63. SNR comparison of image number 5, region number is up to 1000, image size is 900x900.

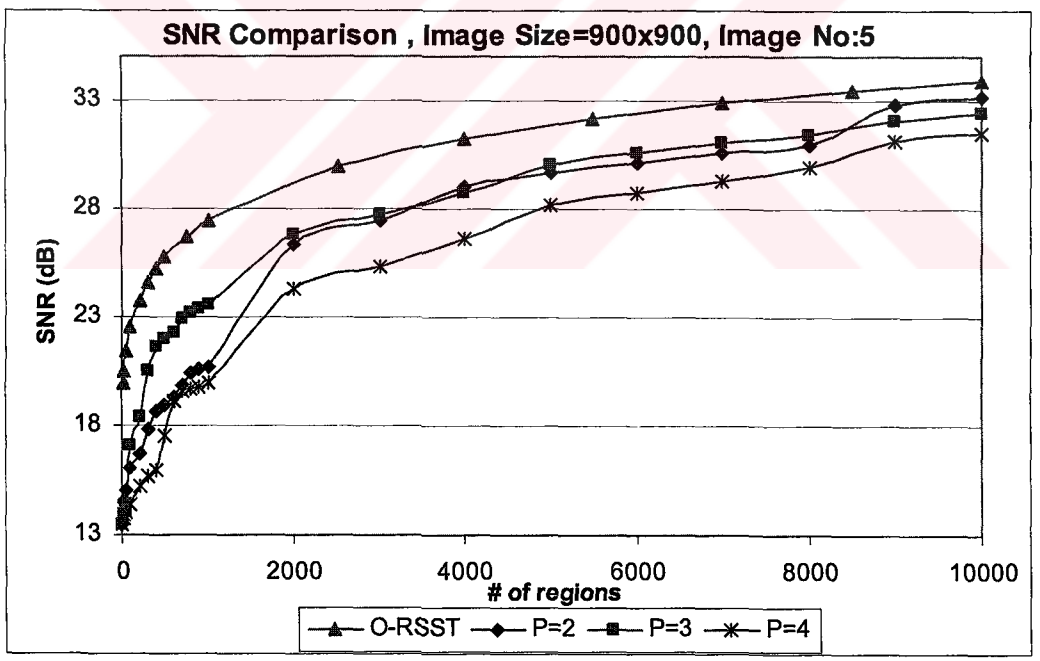
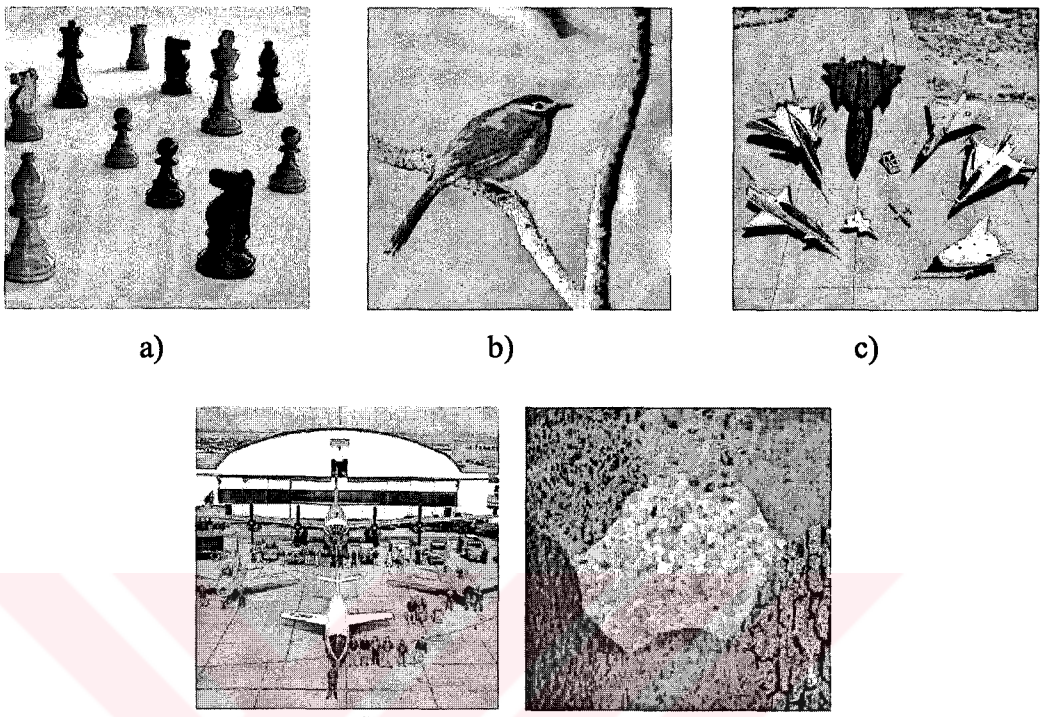


Figure A.64. SNR comparison of image number 1, region number is up to 10,000, image size is 900x900.

APPENDIX F. VISUAL COMPARISON OF SEQUENTIAL ALGORITHMS



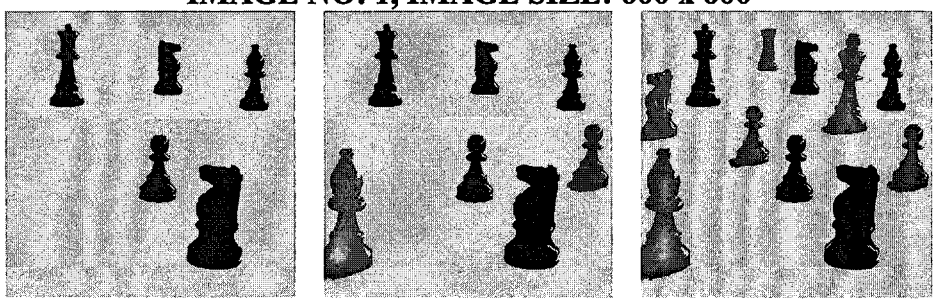
a) b) c)

d) e)

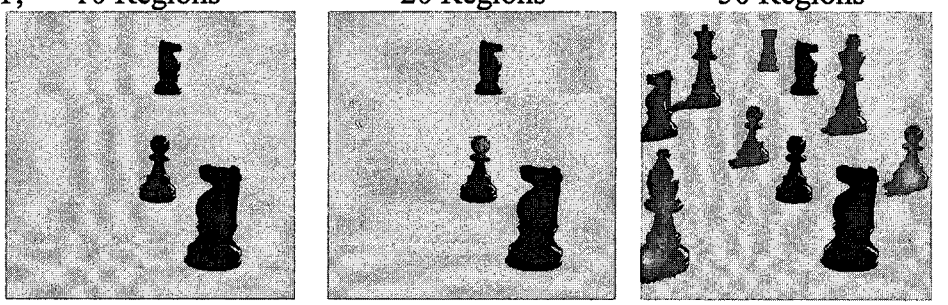
Figure A. 65. Images used throughout the thesis.

Figure A.65 shows the original images used is our experiments. For visual comparison two different image sizes are used. One is 600x600 (360,000 pixels) the other is 900x900 (810,000 pixels).

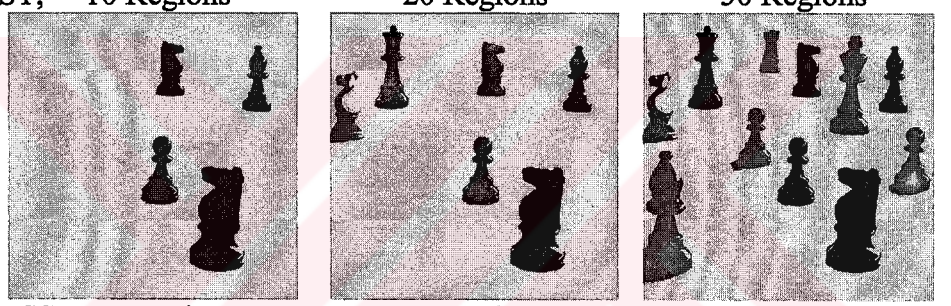
IMAGE NO: 1, IMAGE SIZE: 600 x 600



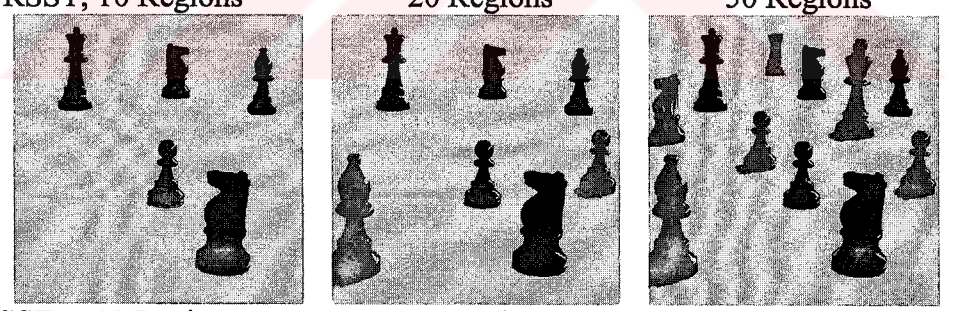
RSST, 10 Regions 20 Regions 50 Regions



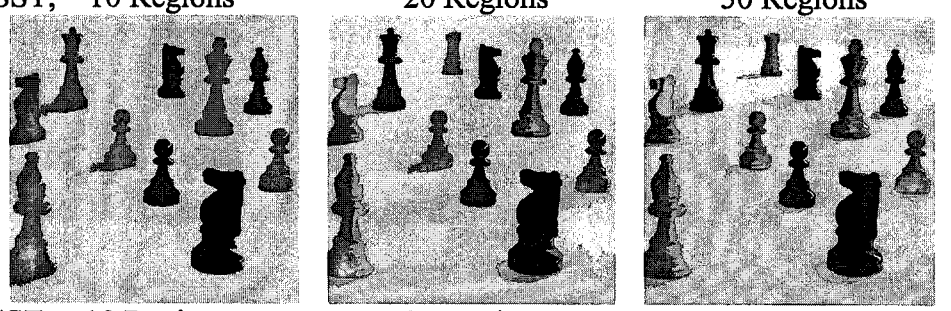
FRSST, 10 Regions 20 Regions 50 Regions



M-FRSST, 10 Regions 20 Regions 50 Regions



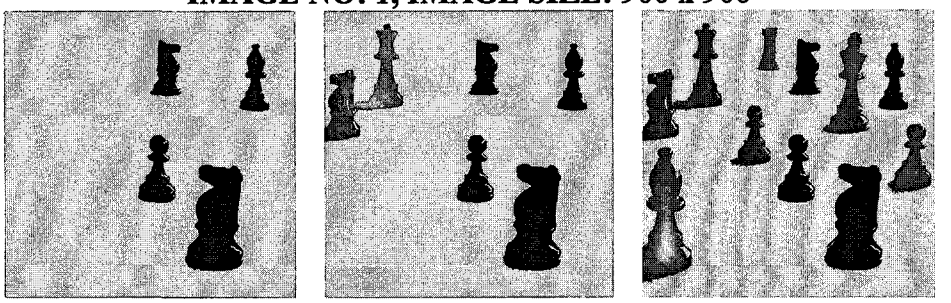
MRSST, 10 Regions 20 Regions 50 Regions



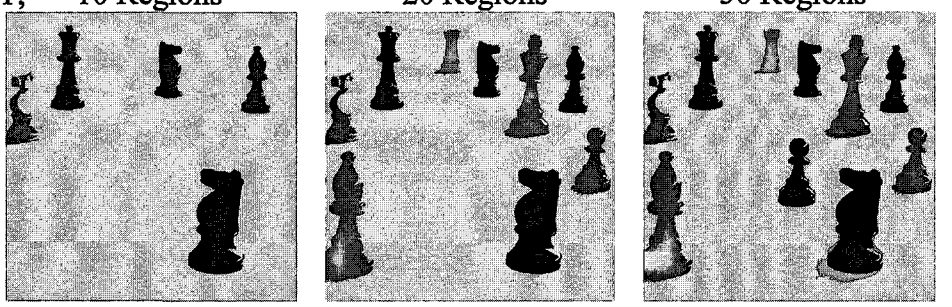
ORSST, 10 Regions 20 Regions 50 Regions

Figure A.66. Segmentation results of Image No: 1 of size 600x600

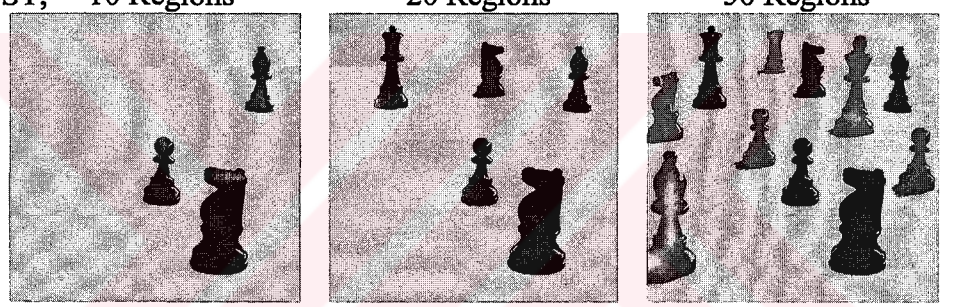
IMAGE NO: 1, IMAGE SIZE: 900 x 900



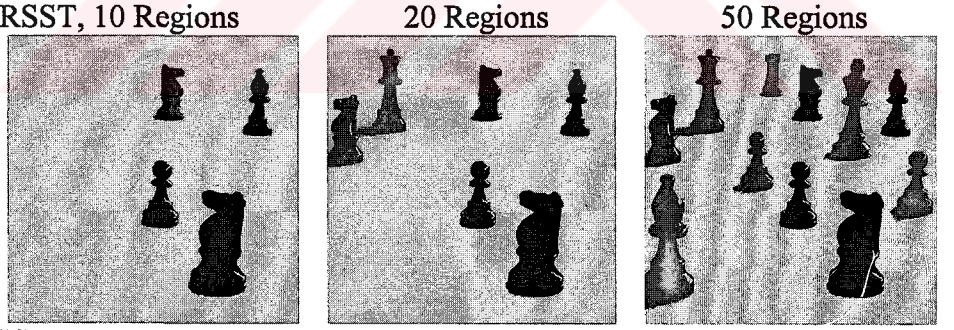
RSST, 10 Regions 20 Regions 50 Regions



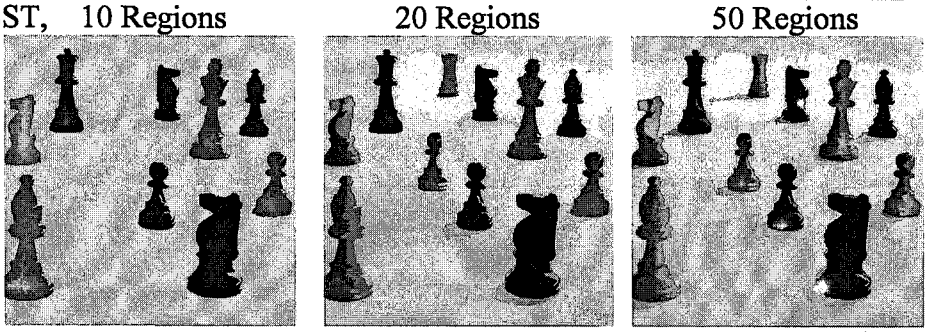
FRSST, 10 Regions 20 Regions 50 Regions



M-FRSST, 10 Regions 20 Regions 50 Regions



MRSST, 10 Regions 20 Regions 50 Regions



ORSST, 10 Regions 20 Regions 50 Regions

Figure A.67. Segmentation results of Image No: 1 of size 900x900

IMAGE NO: 2, IMAGE SIZE: 600 x 600

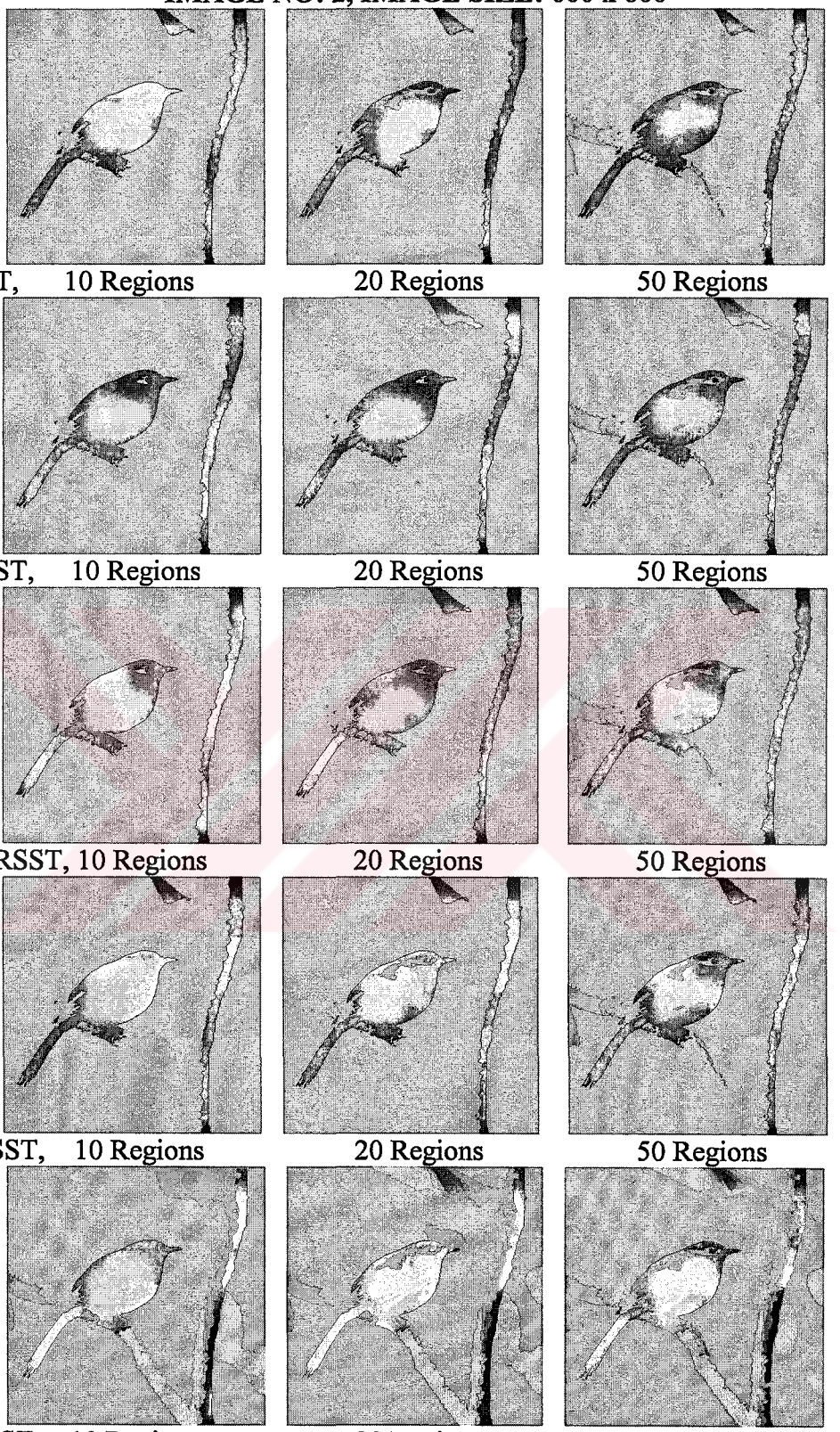
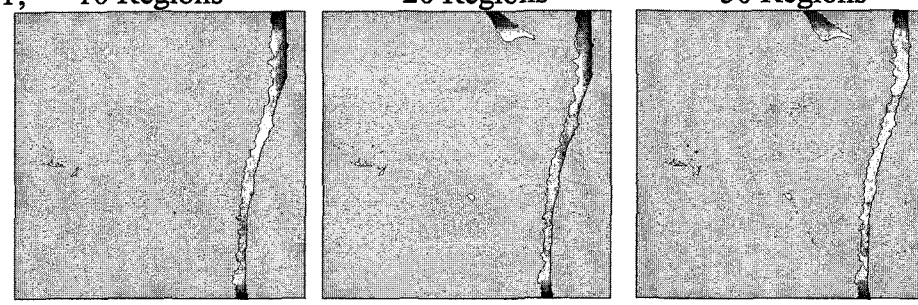


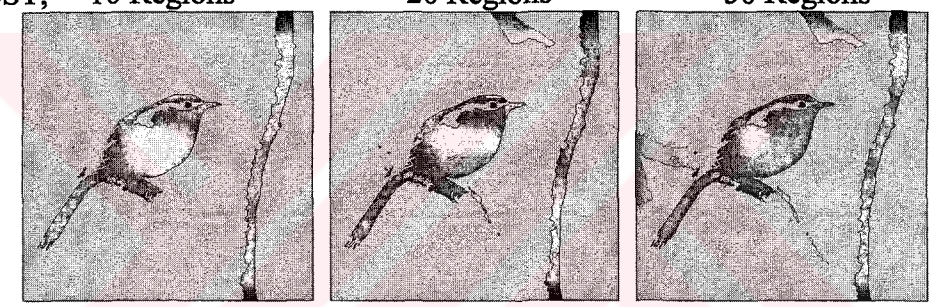
Figure A.68. Segmentation results of Image No: 2 of size 600x600

IMAGE NO: 2, IMAGE SIZE: 900 x 900

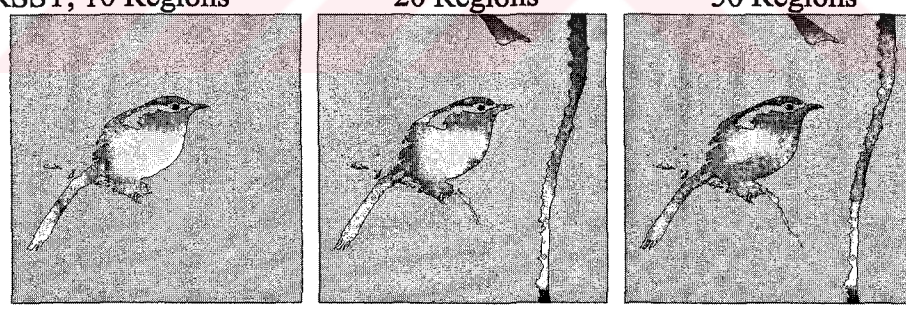
RSST, 10 Regions 20 Regions 50 Regions



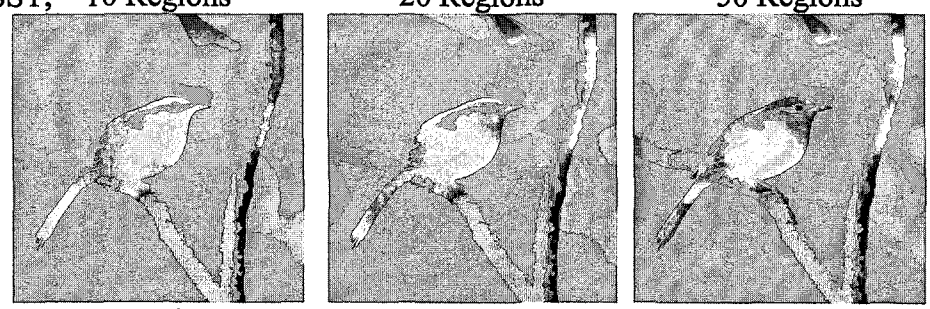
FRSST, 10 Regions 20 Regions 50 Regions



M-FRSST, 10 Regions 20 Regions 50 Regions



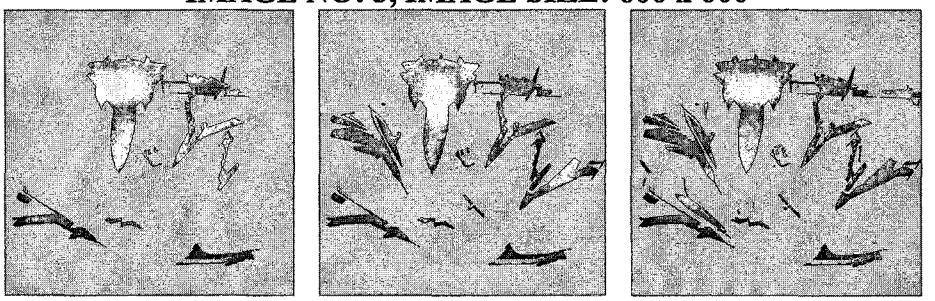
MRSST, 10 Regions 20 Regions 50 Regions



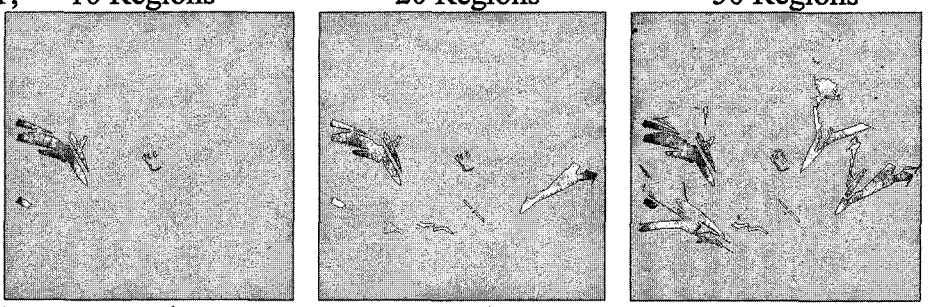
ORSST, 10 Regions 20 Regions 50 Regions

Figure A.69. Segmentation results of Image No: 2 of size 900x900

IMAGE NO: 3, IMAGE SIZE: 600 x 600

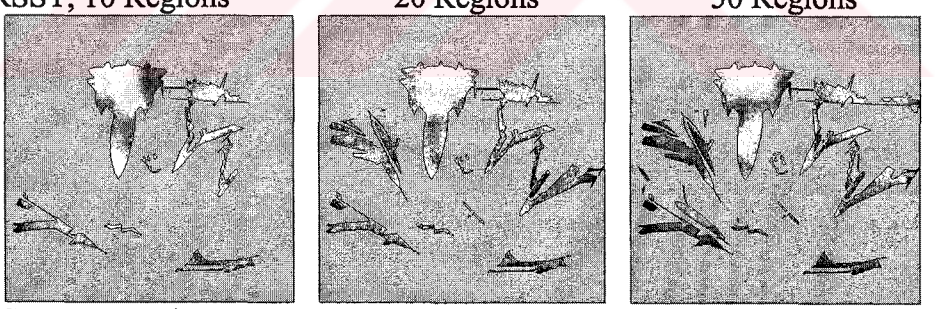


RSST, 10 Regions 20 Regions 50 Regions

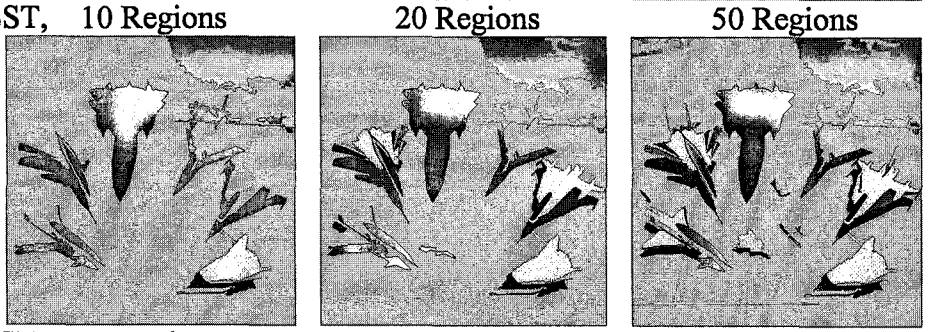


FRSST, 10 Regions 20 Regions 50 Regions

M-FRSST, 10 Regions 20 Regions 50 Regions



MRSST, 10 Regions 20 Regions 50 Regions



ORSST, 10 Regions 20 Regions 50 Regions

Figure A.70. Segmentation results of Image No: 3 of size 600x600

IMAGE NO: 3, IMAGE SIZE: 900 x 900

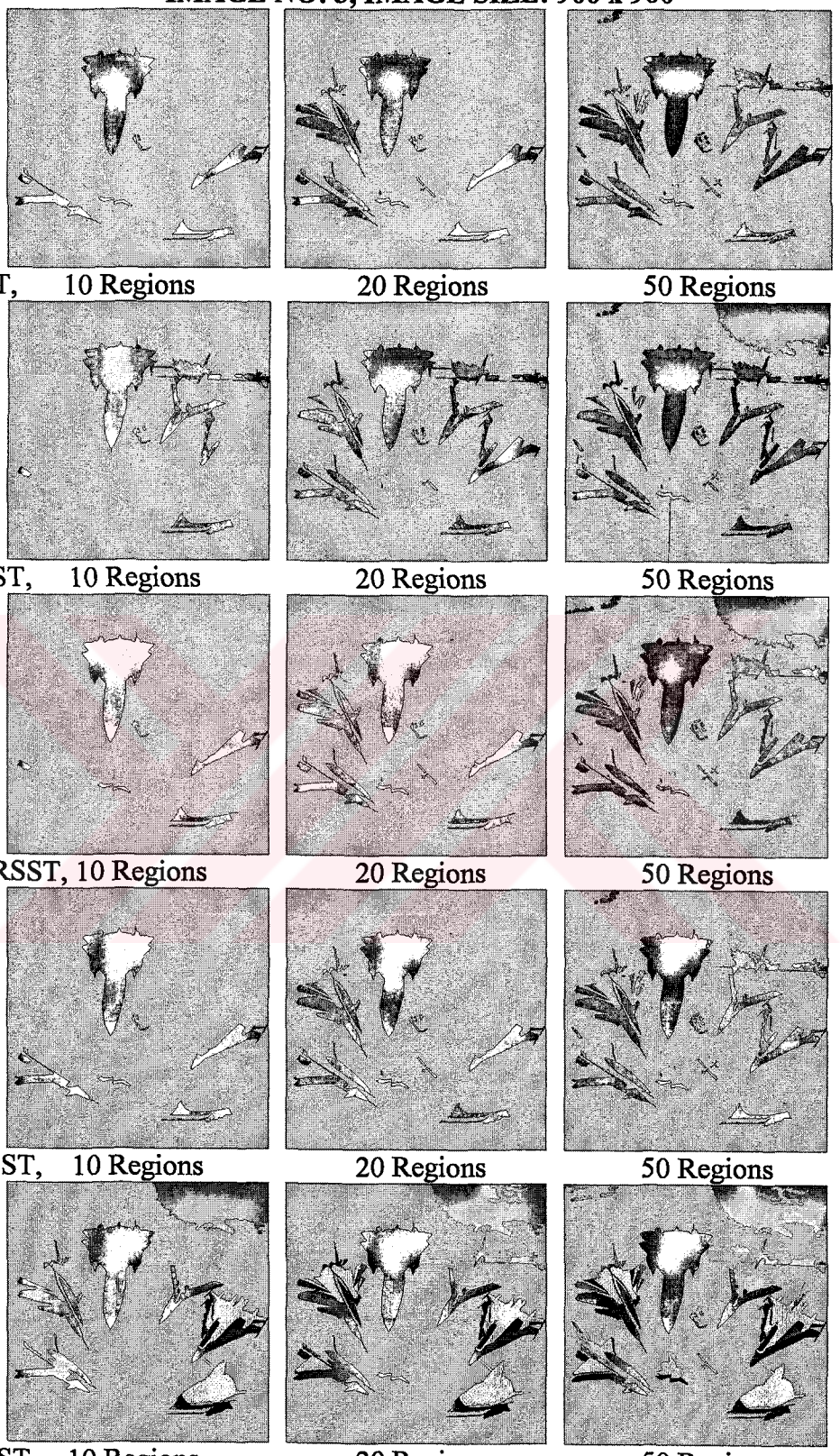
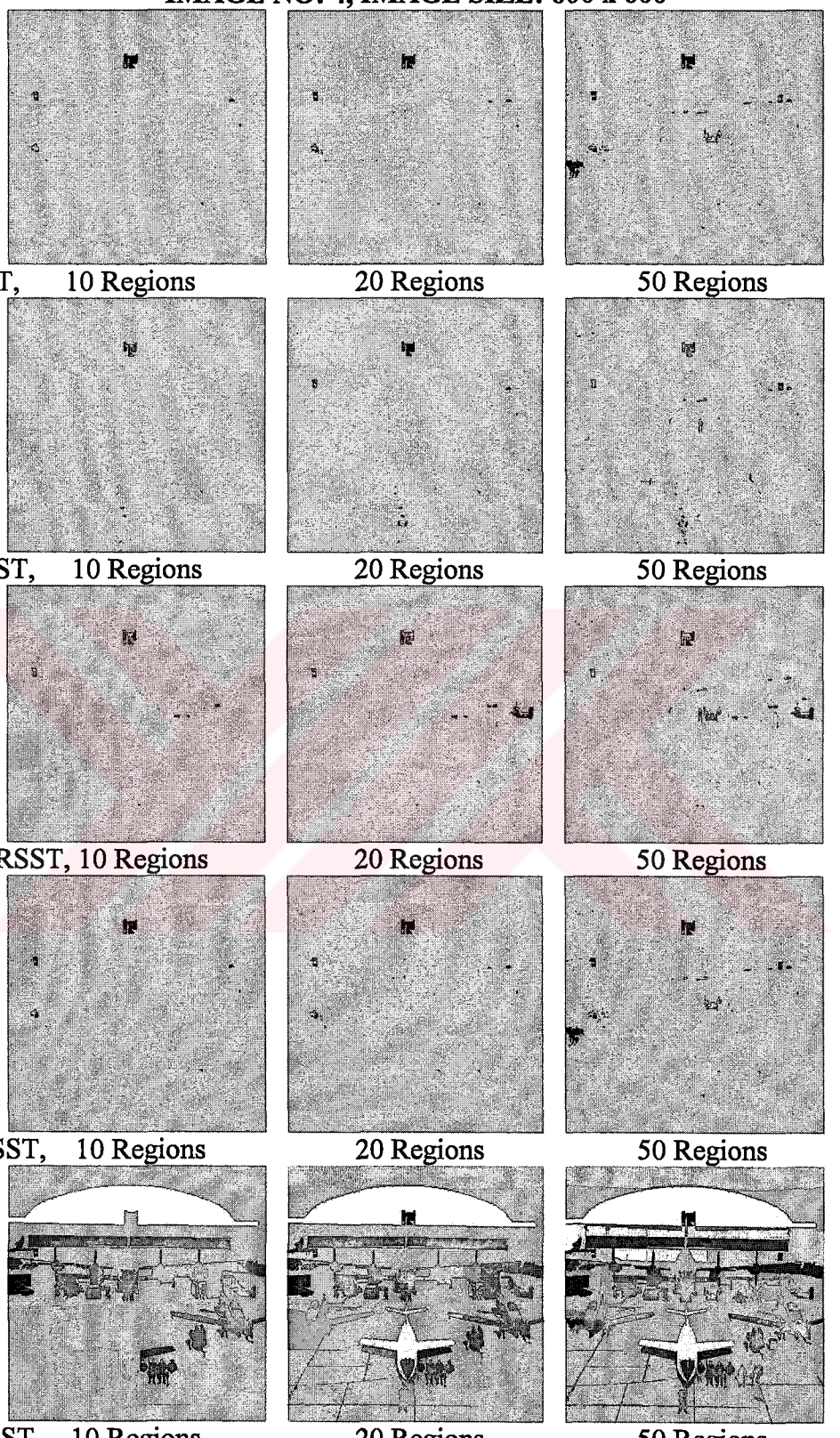


Figure A.71. Segmentation results of Image No: 3 of size 900x900

IMAGE NO: 4, IMAGE SIZE: 600 x 600



RSST, 10 Regions 20 Regions 50 Regions

FRSST, 10 Regions 20 Regions 50 Regions

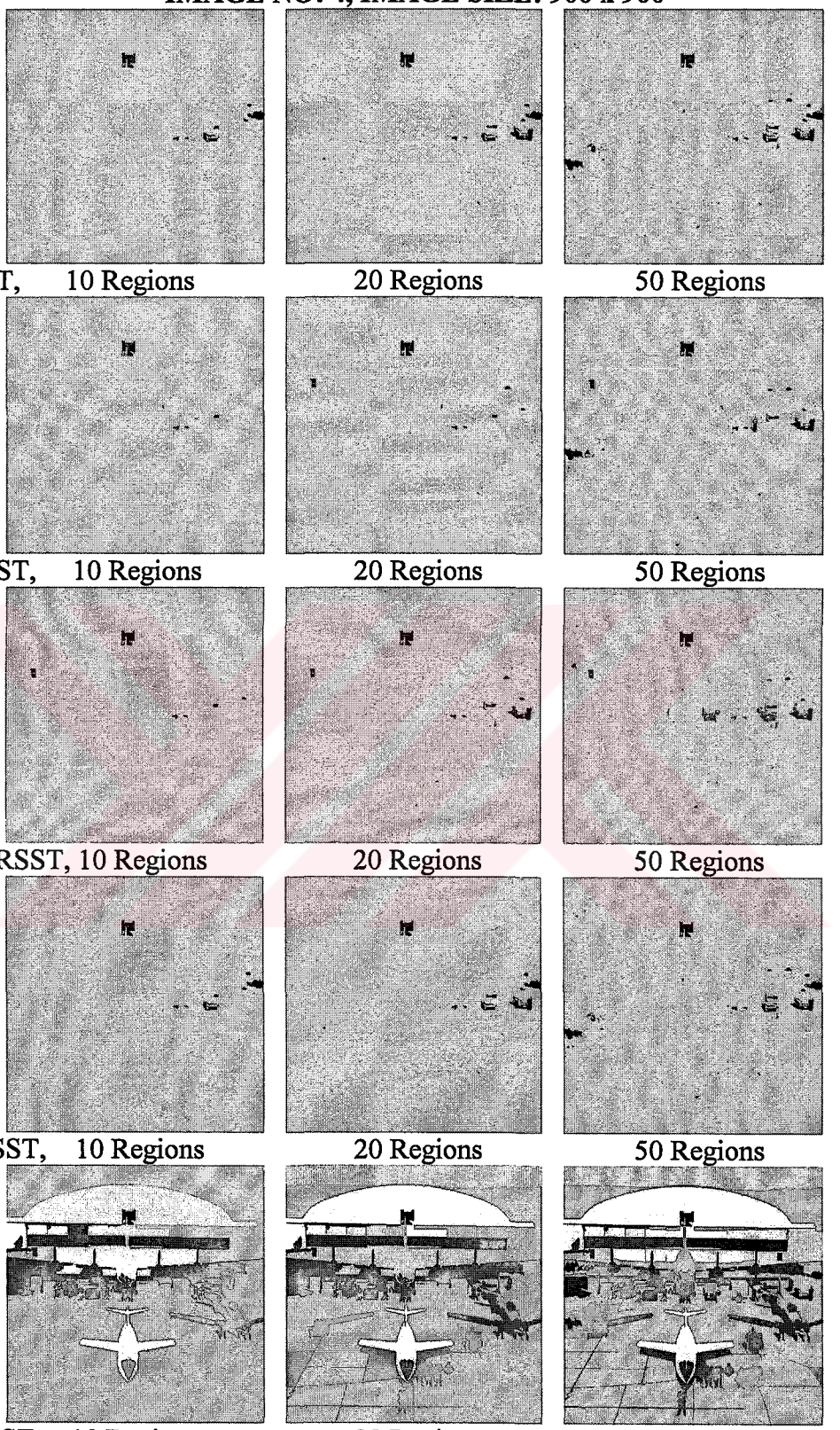
M-FRSST, 10 Regions 20 Regions 50 Regions

MRSST, 10 Regions 20 Regions 50 Regions

ORSST, 10 Regions 20 Regions 50 Regions

Figure A.72. Segmentation results of Image No: 4 of size 600x600

IMAGE NO: 4, IMAGE SIZE: 900 x 900



RSST, 10 Regions 20 Regions 50 Regions

FRSST, 10 Regions 20 Regions 50 Regions

M-FRSST, 10 Regions 20 Regions 50 Regions

MRSST, 10 Regions 20 Regions 50 Regions

ORSST, 10 Regions 20 Regions 50 Regions

Figure A.73. Segmentation results of Image No: 4 of size 900x900

IMAGE NO: 5, IMAGE SIZE: 600 x 600

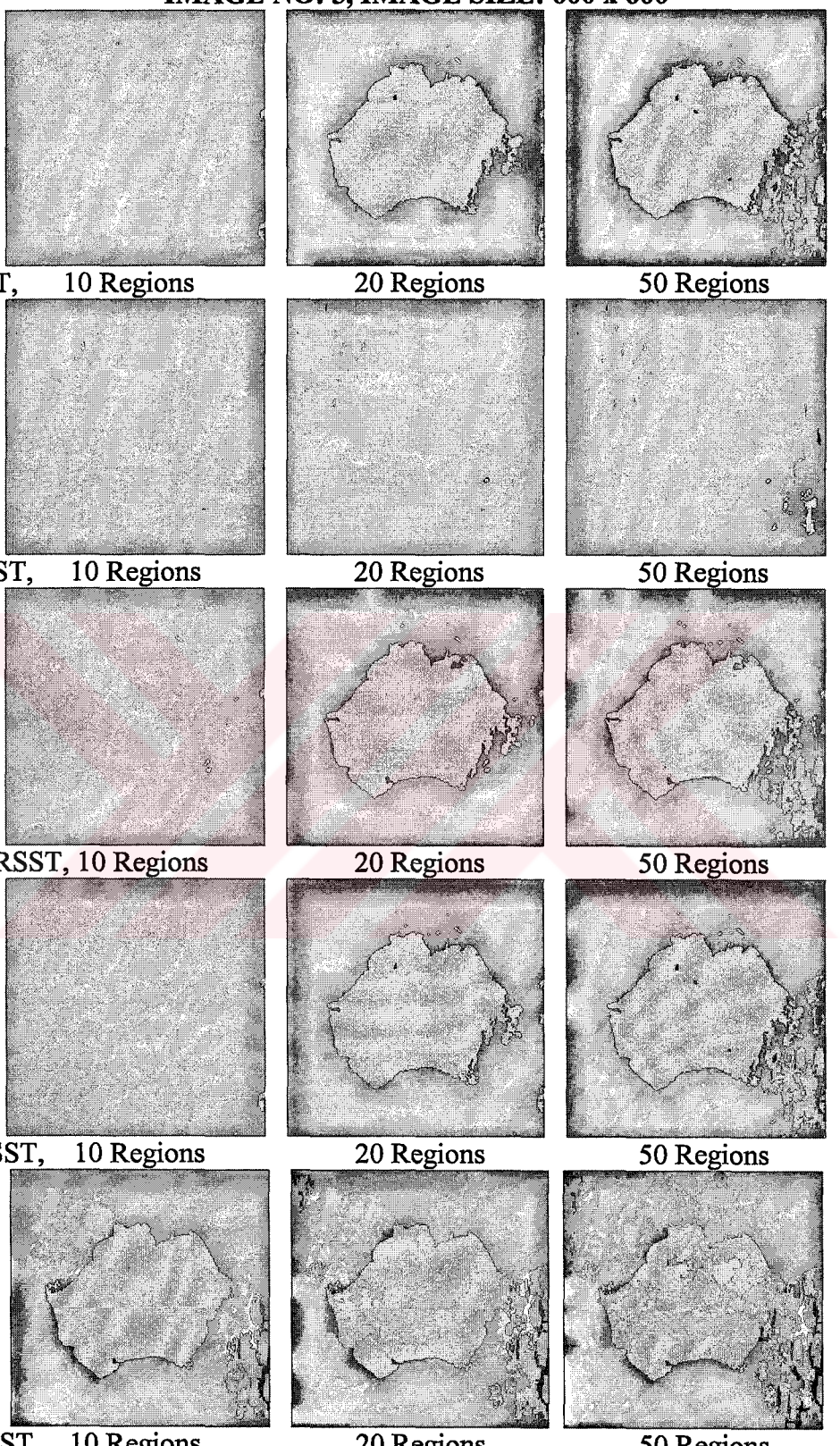


Figure A.74. Segmentation results of Image No: 5 of size 600x600

IMAGE NO: 5, IMAGE SIZE: 900 x 900

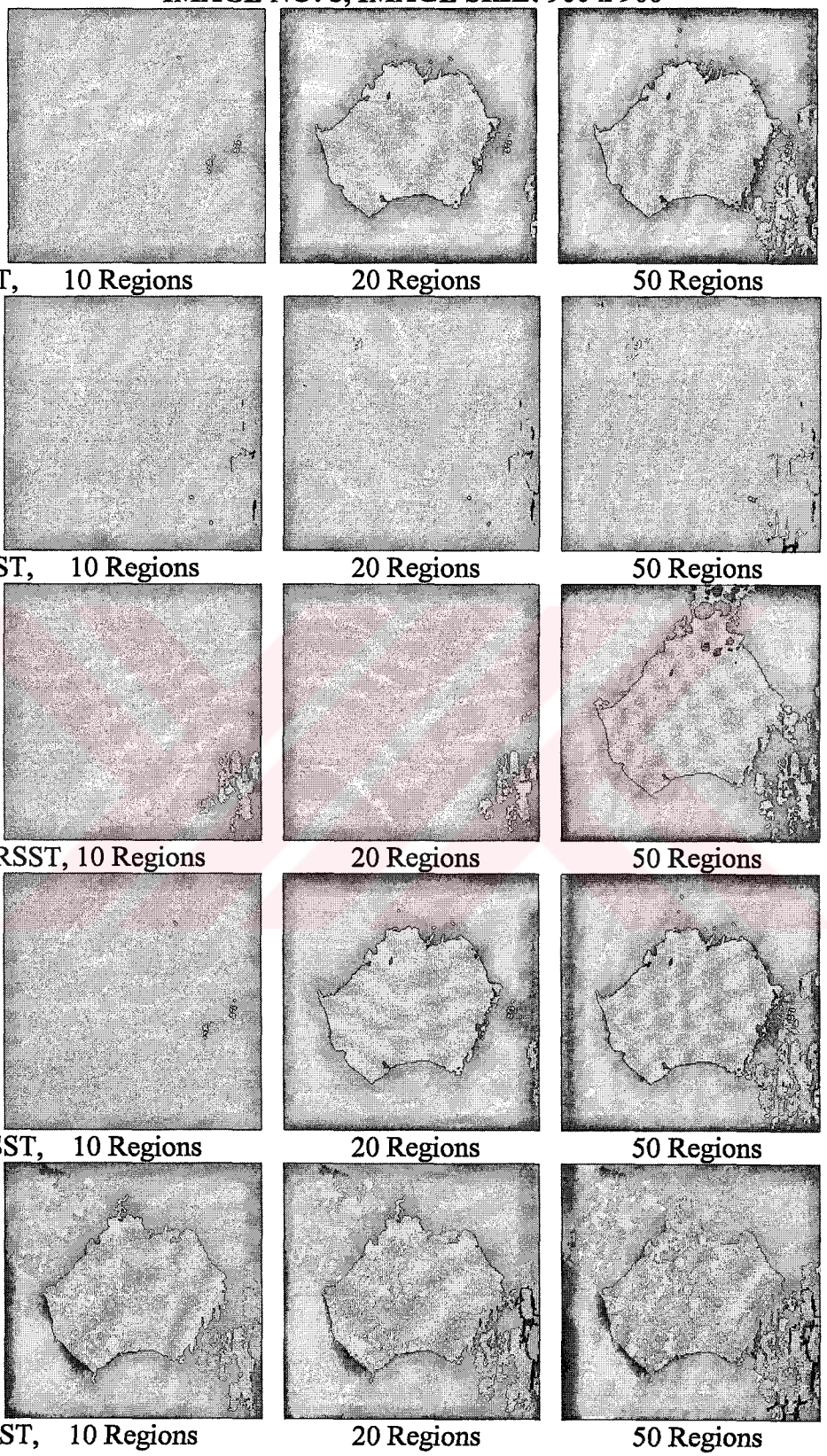
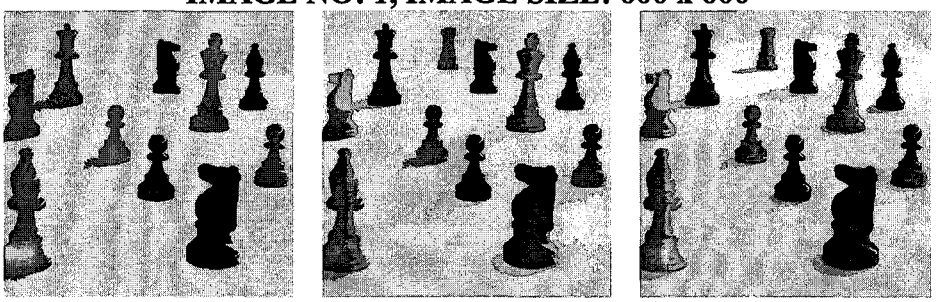


Figure A.75. Segmentation results of Image No: 5 of size 900x900

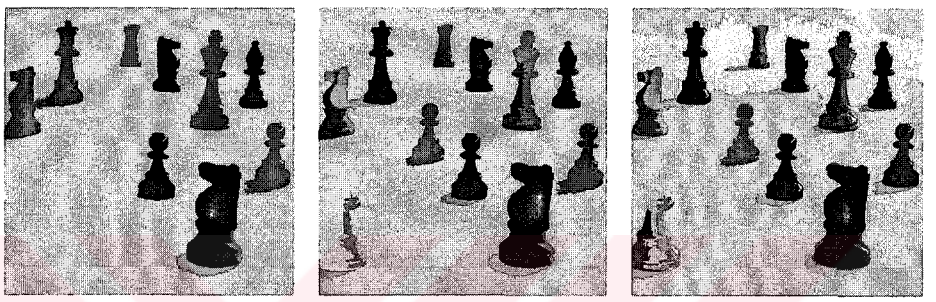
VISUAL COMPARISON OF DISTRIBUTED ALGORITHM

IMAGE NO: 1, IMAGE SIZE: 600 x 600



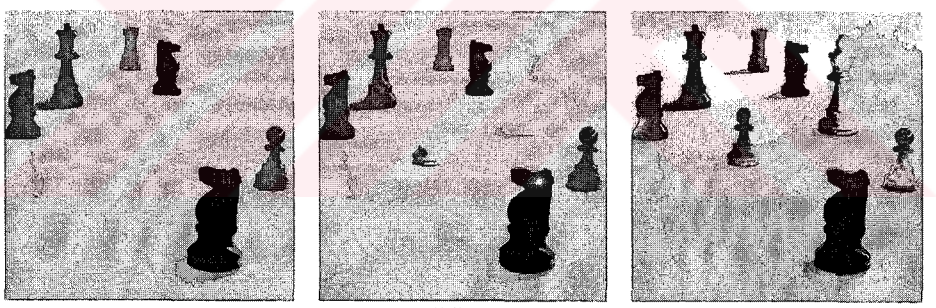
ORSST

10 Regions 20 Regions 50 Regions



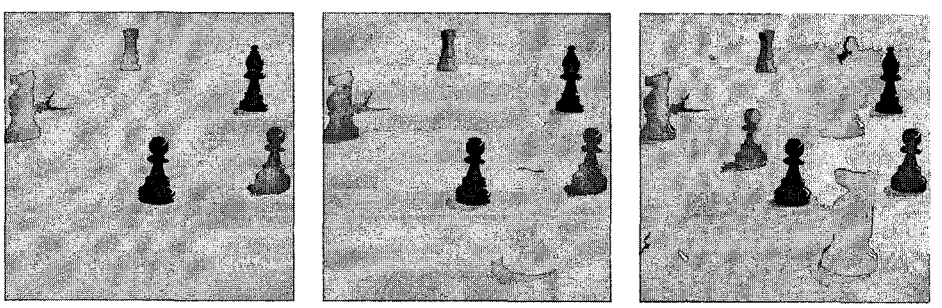
D-RSST, P=2

10 Regions 20 Regions 50 Regions



D-RSST, P=3

10 Regions 20 Regions 50 Regions

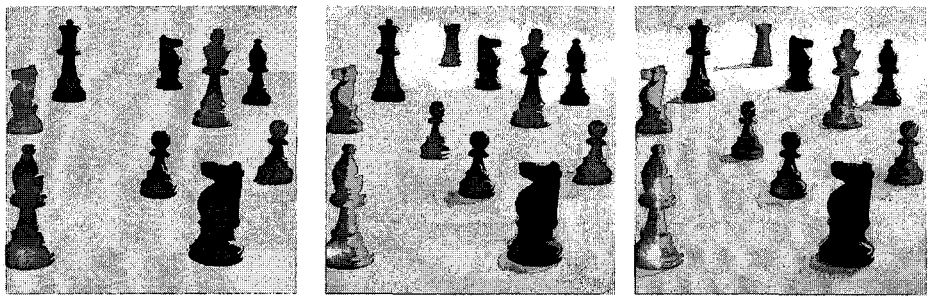


D-RSST, P=4

10 Regions 20 Regions 50 Regions

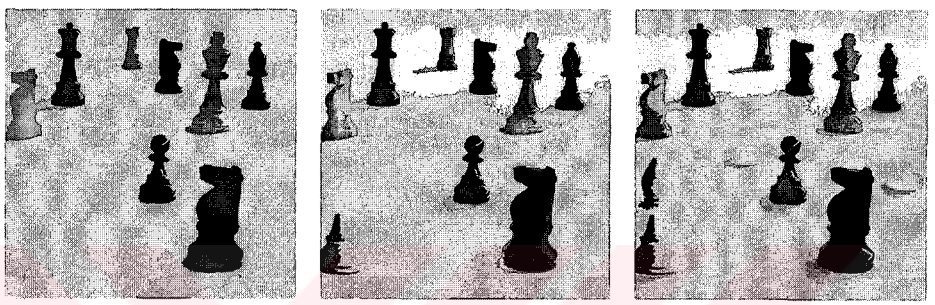
Figure A.76. Segmentation results of Image No: 1 of size 600x600

IMAGE NO: 1, IMAGE SIZE: 900 x 900



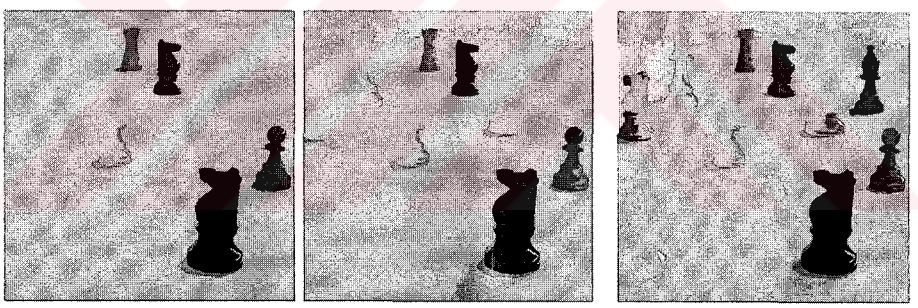
ORSST

10 Regions 20 Regions 50 Regions



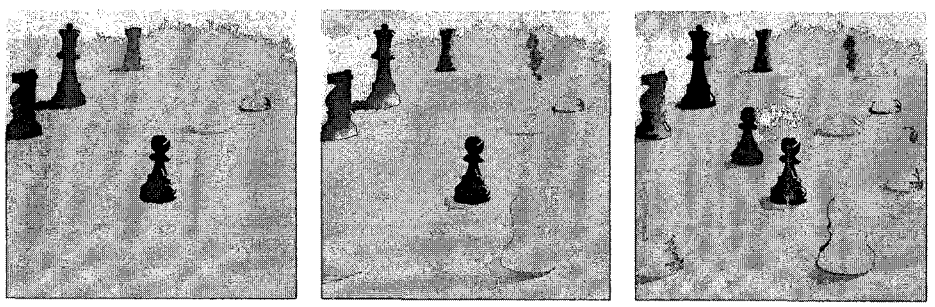
D-RSST, P=2

10 Regions 20 Regions 50 Regions



D-RSST, P=3

10 Regions 20 Regions 50 Regions



D-RSST, P=4

10 Regions 20 Regions 50 Regions

Figure A.77. Segmentation results of Image No: 1 of size 900x900

IMAGE NO: 2, IMAGE SIZE: 600 x 600

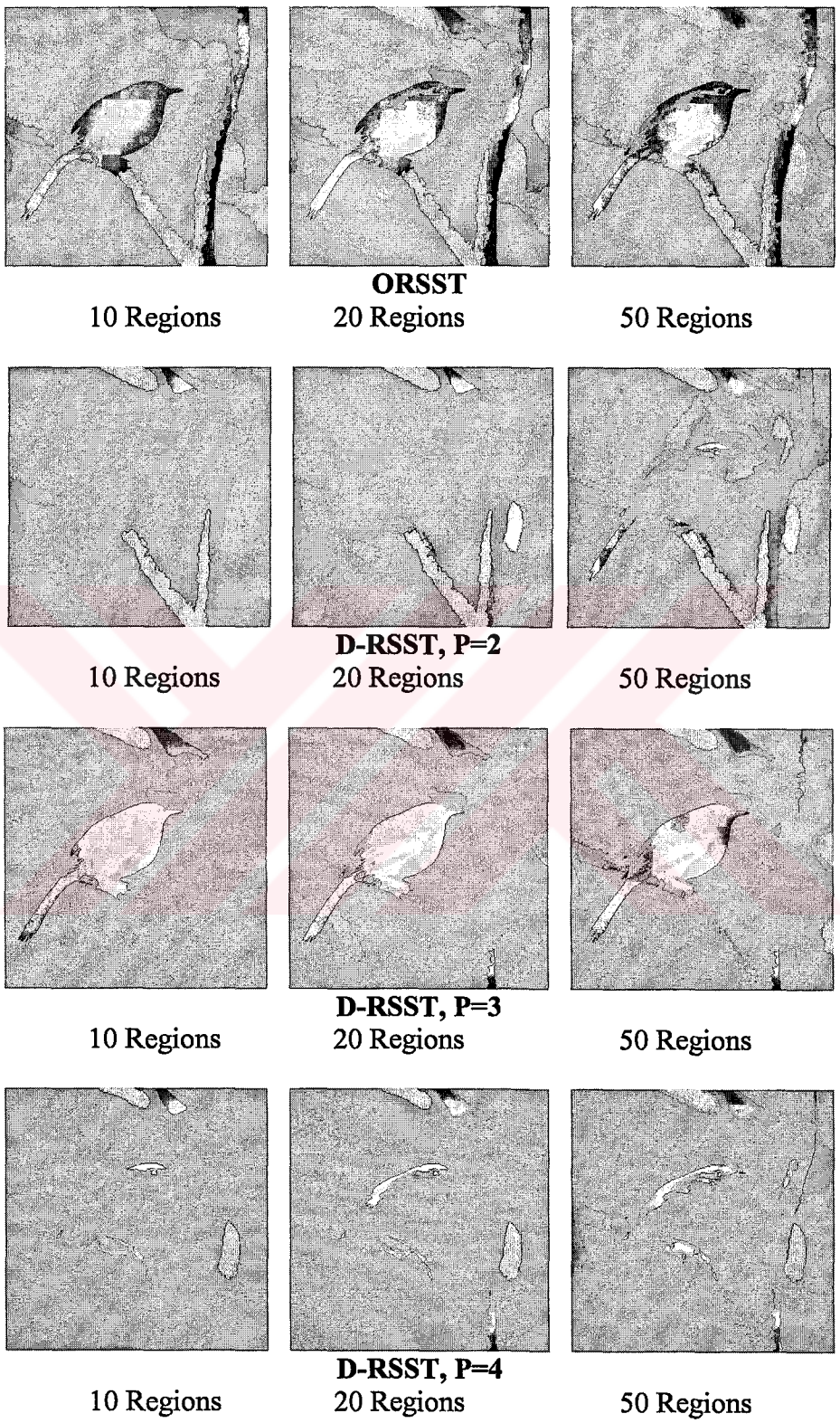


Figure A.78. Segmentation results of Image No:2 of size 600x600

IMAGE NO: 2, IMAGE SIZE: 900 x 900

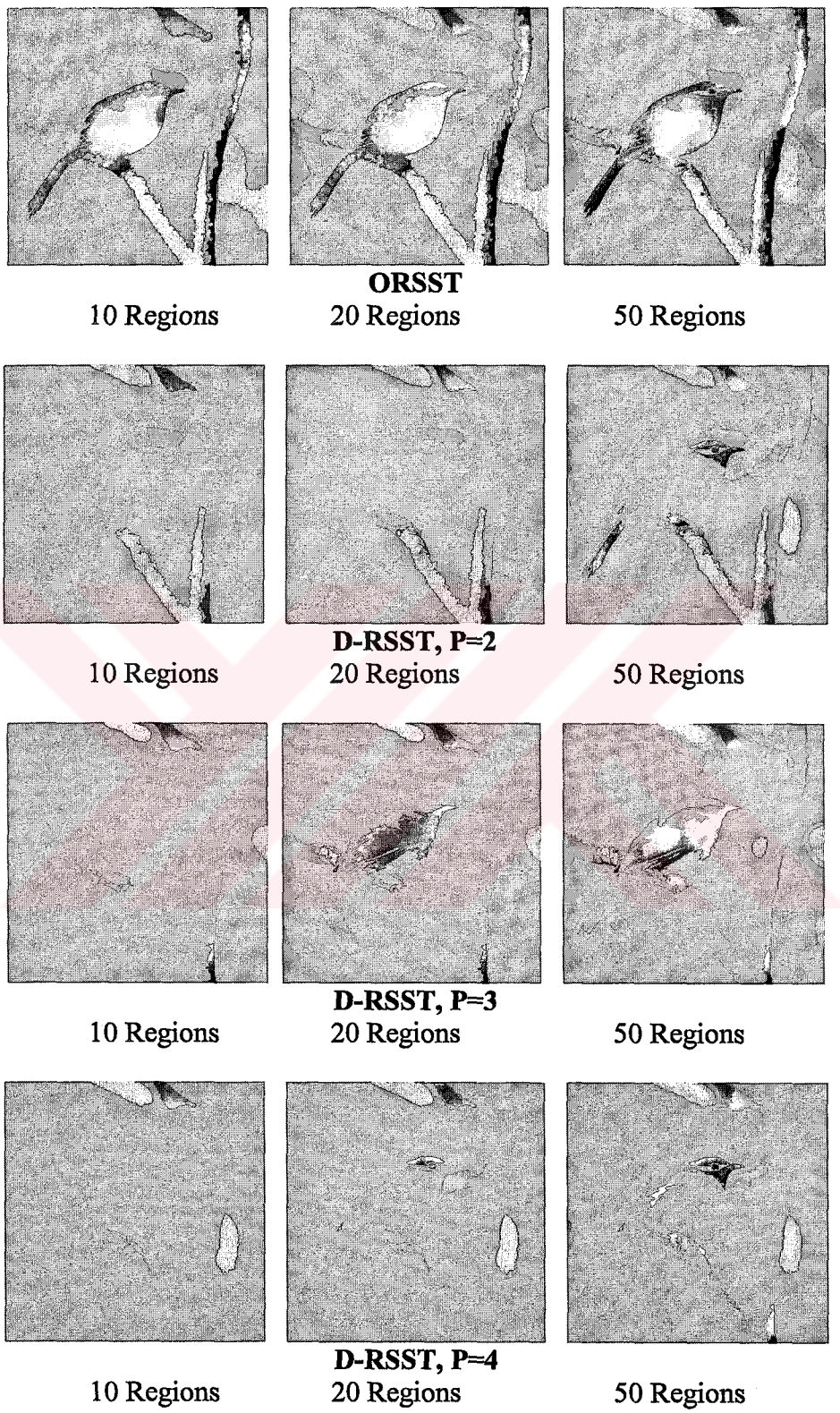
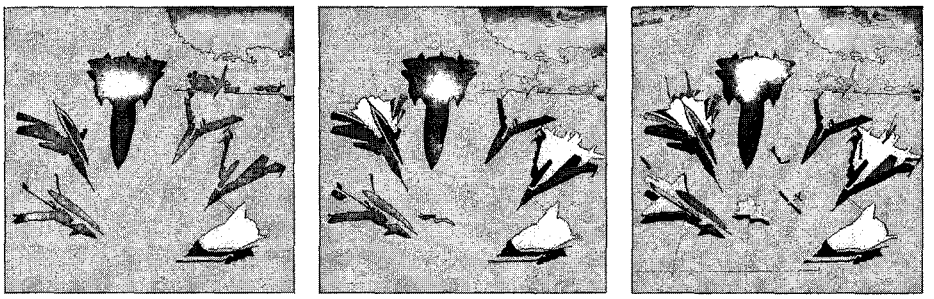


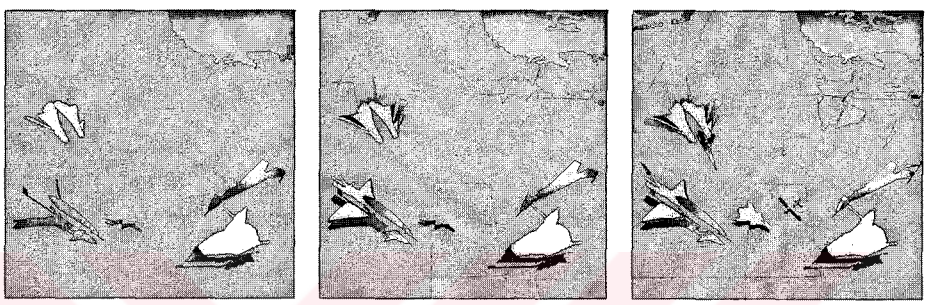
Figure A.79. Segmentation results of Image No:2 of size 900x900

IMAGE NO: 3, IMAGE SIZE: 600 x 600



ORSST

10 Regions 20 Regions 50 Regions

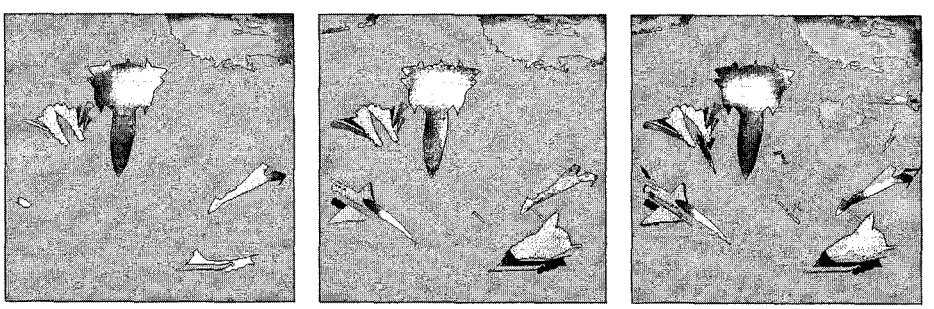


D-RSST, P=2

10 Regions 20 Regions 50 Regions

D-RSST, P=3

10 Regions 20 Regions 50 Regions

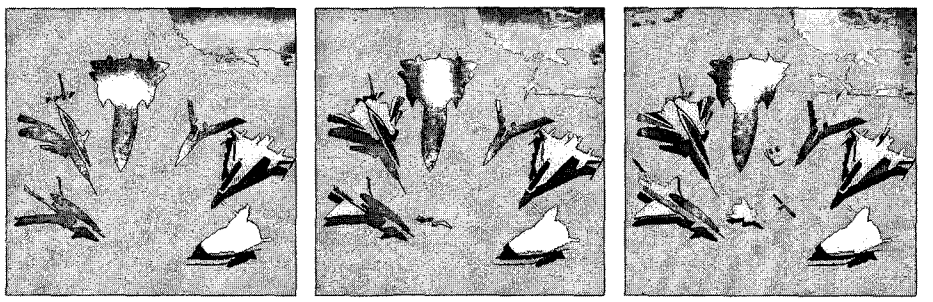


D-RSST, P=4

10 Regions 20 Regions 50 Regions

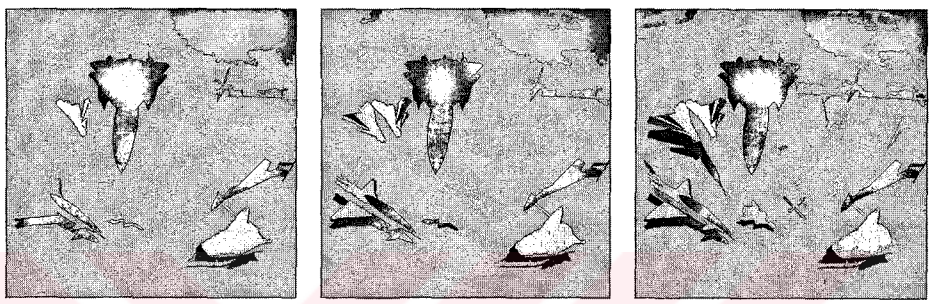
Figure A.80. Segmentation results of Image No: 3 of size 600x600

IMAGE NO: 3, IMAGE SIZE: 900 x 900



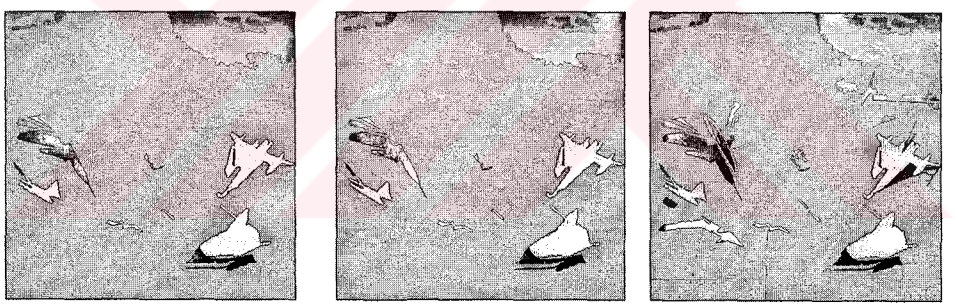
ORSST

10 Regions 20 Regions 50 Regions



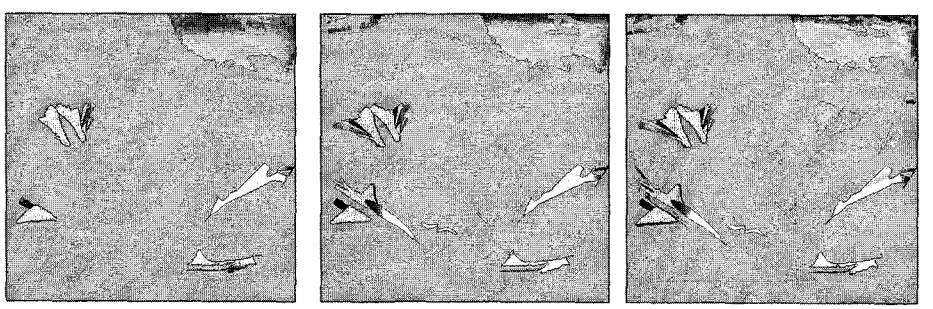
D-RSST, P=2

10 Regions 20 Regions 50 Regions



D-RSST, P=3

10 Regions 20 Regions 50 Regions

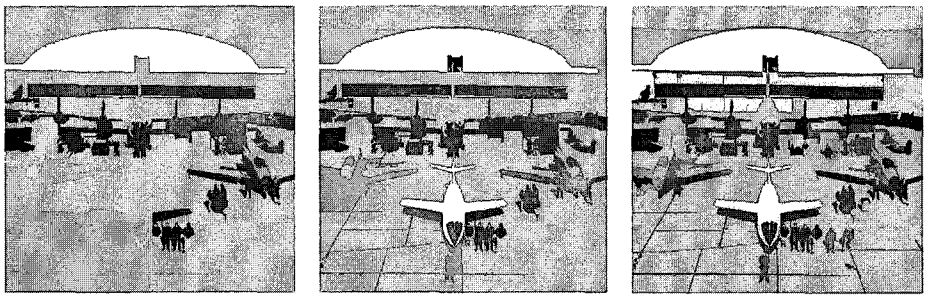


D-RSST, P=4

10 Regions 20 Regions 50 Regions

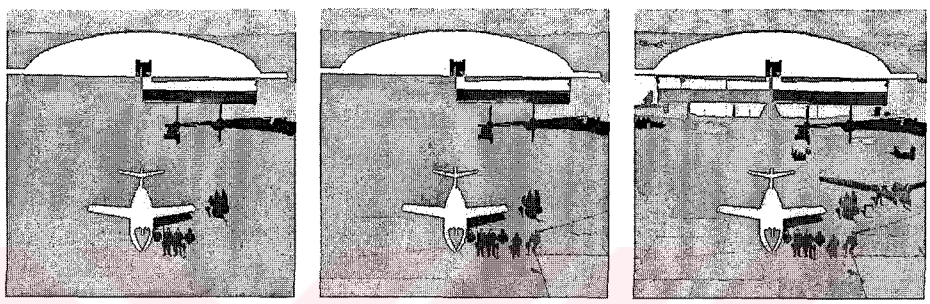
Figure A.81. Segmentation results of Image No: 3 of size 900x900

IMAGE NO: 4, IMAGE SIZE: 600 x 600



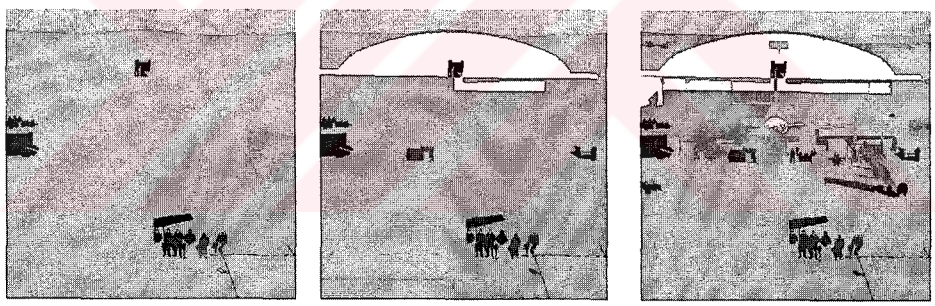
ORSST

10 Regions 20 Regions 50 Regions



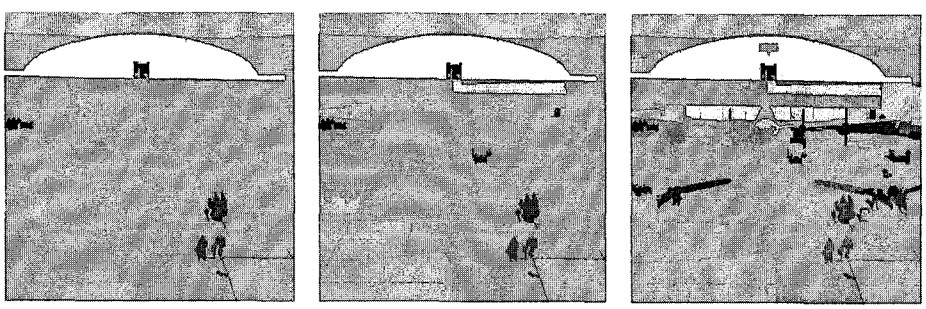
D-RSST, P=2

10 Regions 20 Regions 50 Regions



D-RSST, P=3

10 Regions 20 Regions 50 Regions

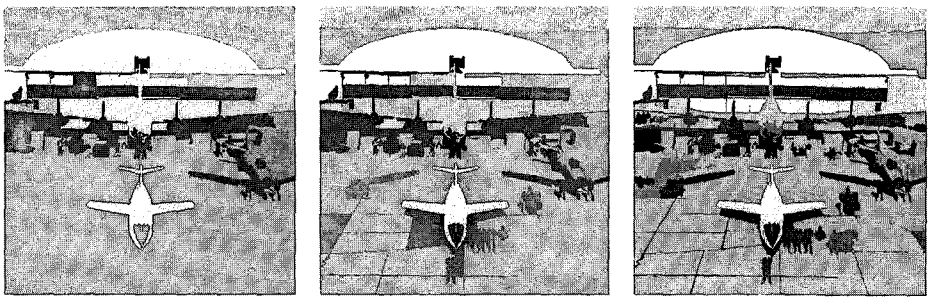


D-RSST, P=4

10 Regions 20 Regions 50 Regions

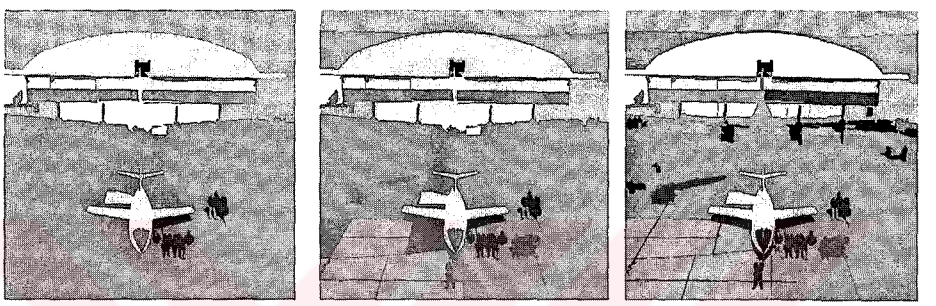
Figure A.82. Segmentation results of Image No: 4 of size 600x600

IMAGE NO: 4, IMAGE SIZE: 900 x 900



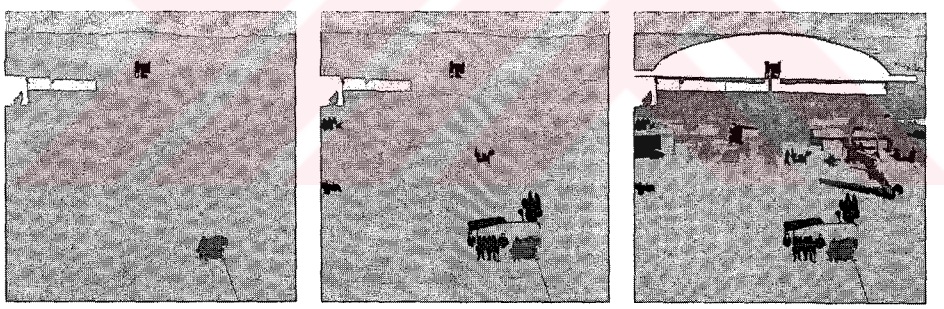
ORSST

10 Regions 20 Regions 50 Regions



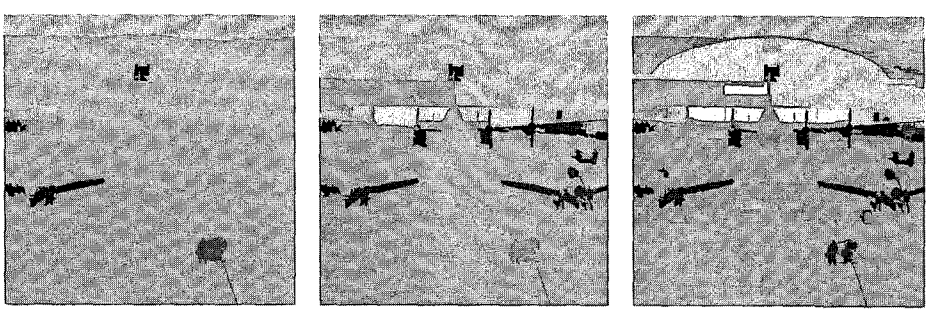
D-RSST, P=2

10 Regions 20 Regions 50 Regions



D-RSST, P=3

10 Regions 20 Regions 50 Regions



D-RSST, P=4

10 Regions 20 Regions 50 Regions

Figure A.83. Segmentation results of Image No: 4 of size 900x900

IMAGE NO: 5, IMAGE SIZE: 600 x 600

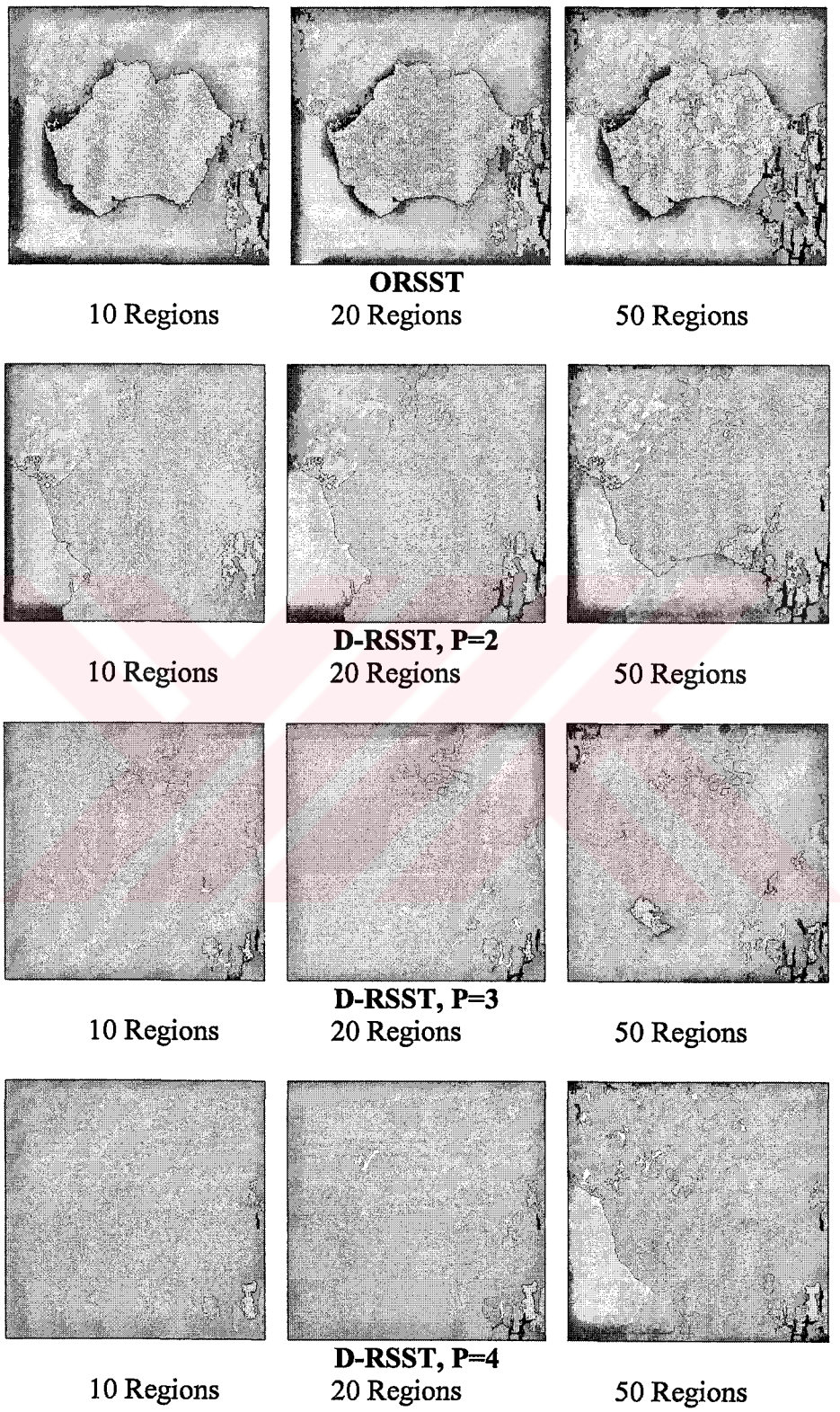


Figure A.84. Segmentation results of Image No: 5 of size 600x600

IMAGE NO: 5, IMAGE SIZE: 900 x 900

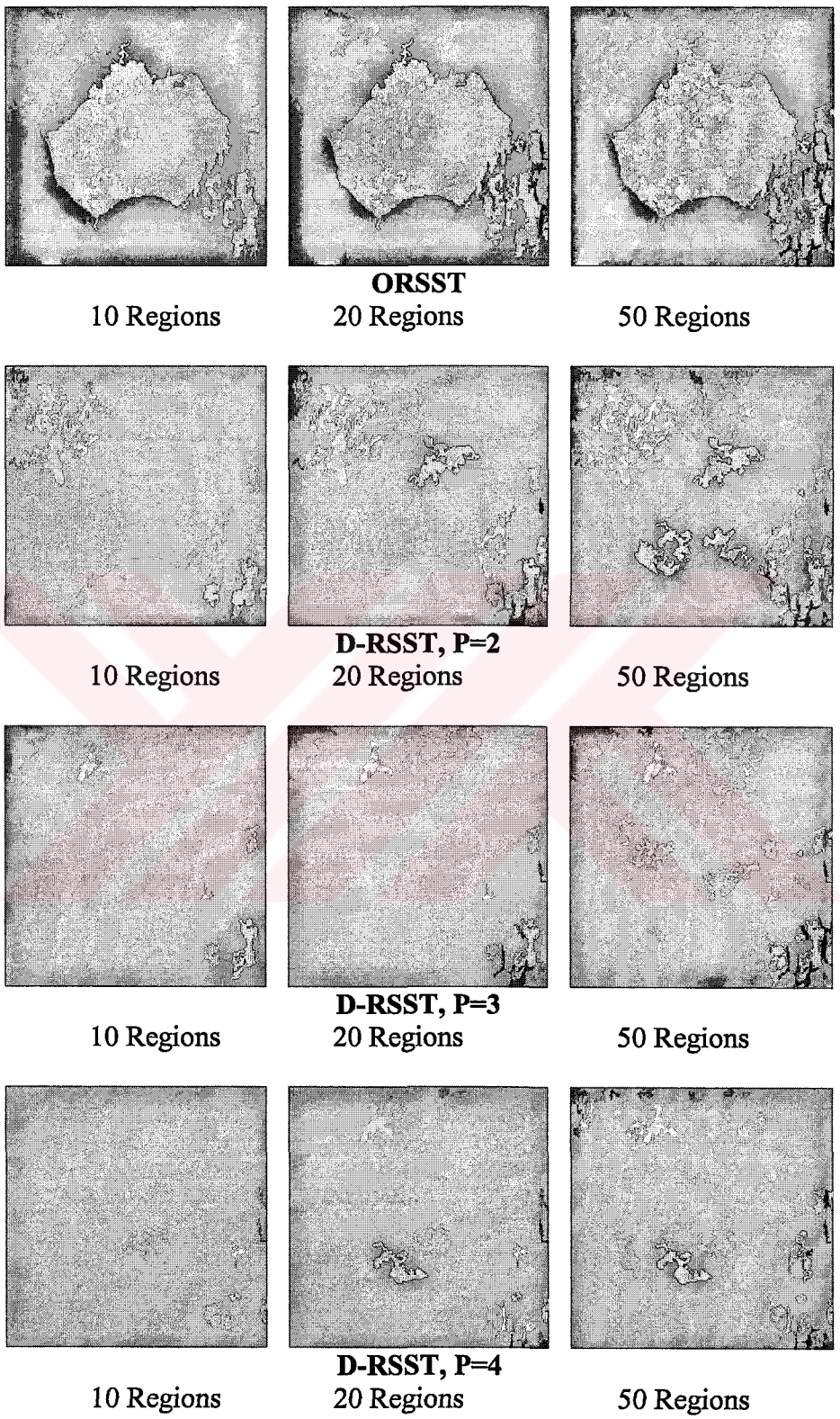


Figure A.85. Segmentation results of Image No: 5 of size 900x900