1999

Genetic algorithm and tabu search approaches to quantization for DCT-based image compression

Michael Champion

Rochester Institute of Technology
Computer Science Department

Genetic Algorithm and Tabu Search Approaches to Quantization for DCT-based Image Compression

by Michael S. Champion
A thesis, submitted to
The Faculty of the Computer Science Department
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Approved by:

Dr. Roger Gaborski
Rochester Institute of Technology

Dr. Peter Anderson
Rochester Institute of Technology

Dr. Fereydoun Kazemian
Rochester Institute of Technology

January 25, 1999

Rochester Institute of Technology
Computer Science Department

Genetic Algorithm and Tabu Search Approaches to Quantization for DCT-based Image Compression

Signed:

_____

Michael Champion

January 25, 1999

Masters Thesis (Winter 1998/1999)

GENETIC ALGORITHM AND TABU SEARCH APPROACHES TO QUANTIZATION FOR DCT-BASED IMAGE COMPRESSION

Michael S. Champion (e-mail: msc0686@cs.rit.edu)

Rochester Institute of Technology

January 28, 1999

1 Abstract

Today there are several formal and experimental methods for image compression, some of which have grown to be incorporated into the Joint Photographers Experts Group (JPEG) standard. Of course, many compression algorithms are still used only for experimentation mainly due to various performance issues. Lack of speed while compressing or expanding an image, poor compression rate, and poor image quality after expansion are a few of the most popular reasons for skepticism about a particular compression algorithm.

This paper discusses current methods used for image compression. It also gives a detailed explanation of the discrete cosine transform (DCT), used by JPEG, and the efforts that have recently been made to optimize related algorithms. Some interesting articles regarding possible compression enhancements will be noted, and in association with these methods a new implementation of a JPEG-like image coding algorithm will be outlined. This new technique involves adapting between one and sixteen quantization tables for a specific image using either a genetic algorithm (GA) or tabu search (TS) approach. First, a few schemes including pixel neighborhood and

Kohonen self-organizing map (SOM) algorithms will be examined to find their effectiveness at classifying blocks of edge-detected image data. Next, the GA and TS algorithms will be tested to determine their effectiveness at finding the optimum quantization table(s) for a whole image. A comparison of the techniques utilized will be thoroughly explored.

2 Acknowledgments

I would like to thank David Foos, Lori Barski, Dr. Susan Young, and Dr. Roger Gaborski of Eastman Kodak's Health Imaging Research Labs (HIRL) for sharing their knowledge of image compression and image understanding. Dr. Gaborski also provided insights concerning the Kohonen SOM and edge-detection algorithms, in addition to supplying current literature about these subjects. I would also like to thank Dr. Peter Anderson of the Rochester Institute of Technology (RIT) Department of Computer Science for access to his generic genetic algorithm code, as well as lending his knowledge of image block classification.

3 Table of Contents

4 Introduction and Background

Currently there are many ways in which images can be compressed for leaner storage or faster data transmission, and they all can be defined in one of two classes: lossless and lossy. Lossless methods are able to restore the data identical to the original, normally at the expense of a worse rate of compression. However, lossy techniques compromise quality for a lower transmission rate by quantizing to minimize irrelevant information. This method is considered desirable if the decrease in compressed size is relatively larger than the amount of loss in quality. Transform coding is usually used to minimize output values of the transformation used (e.g. DCT), and a lossy coding compression strategy exploiting this trait is often used because all natural measurements contain some sort of noise ([COMP97]). There are many examples of both lossless and lossy types of image coding, many of which have been used in nearly-lossless algorithms (see Appendix A, Figures 2 and 3).

Most of the focus of this paper concerns the JPEG image compression algorithm. In JPEG, the image is initially transformed from red-green-blue (RGB) to video-based encoding in order to separate the greyscale image (luminance) from the color information (chrominance). Each portion is compacted separately because it is possible to lose more color than greyscale during compression. Next, subregions are processed with DCT using real number values, but based upon the complex numbers of the Fast Fourier Transform (FFT). Finally, additional compression is done by truncating the precision of terms and using a coding scheme that avoids repetition of characters (see Appendix A, Figure 4). DCT fits in well with the desire to perform forward and reverse transforms quickly with minimal loss of image quality. It can encode image information that can be reconstructed within an arithmetic precision of the computer. This precision is generally better than the precision of an original image sensor or analog-to-digital converter. However, the human eye uses greyscale edges to detect boundaries, allowing colors to bleed across these boundaries without confusion. JPEG should take more into account these edge factors through the implementation of edge-specific quantization matrices which could be classified using a method such as the Kohonen self-organizing map (SOM) algorithm or Connected Pixel Neighborhood algorithms.

4.1 Problem Statement

This purpose of this paper is to describe a new adaptive compression process whereby multiple quantization tables are used to code an image. Quantization is the step which allows a continuous amplitude analog signal to be placed in the discrete amplitude representation of a digital computer. The classification of different types of quantization tables are attempted using a Kohonen SOM, and then compared with two simpler techniques. In the case of the Kohonen SOM, edge-detected blocks of the image are fed as training vectors into the neural network until convergence on a set of weights is achieved. The easier methods simply use the edge-detected data alone or they try to perimeterize the edges using connected pixel neighborhoods. Once the classification is complete, both the GA and TS algorithms are compared as they each try to find the optimum quantization table for the different types of classified blocks. The DCT is used to transform each block with its assigned quantization table during this testing period. An evaluation of this entire process is explored in detail based on its application to several different images and some images of the same type. The results should indicate that the methods just described provide a means to effectively assist JPEG image compression.

The problem with enhancing image compression obviously lies in the fact that there are many different formats which still could use some optimization. Many of the methods shown in Figures 2 and 3 of Appendix A are complex, but with enough research each of them stand the chance of being improved using an abundance of optimization techniques. Relatively simple lossy compression algorithms, like DCT and DWT (discrete wavelet transform), are linear operations generating a data structure that has $log_2 N$ segments of various lengths, where N is the length of one side of an image. Usually the algorithm fills the structure and transforms it into a changed data vector 2^N in length. Both transforms are relatively easy to code, and they can be viewed simply as a rotation to a different domain in function space. For DCT the domain is made up only of cosine (or sine) functions, and for DWT the basis functions are more complex ones called wavelets ([GRAP95]).

Analyzing (or mother) wavelet functions have their frequencies localized in space so that functions and operators using them will be "sparse" when transformed into the wavelet domain. This sparseness aids feature detection, noise removal, and image compression. Wavelets provide an easy choice for a defining coefficient in a

given wavelet system to be adapted for a given problem. They also process data at different scales of resolution. The fast DWT, like the FFT used for DCT, is capable of factoring its transform into a product of few sparse matrices by using self-similarity properties. Wavelet packets, or linear combinations of wavelets, are computed by a recursive algorithm which makes each new packet the root of an analysis tree (see Appendix A, Figure 5). This tree is used to retain orthogonality, smoothness, and localization properties of parent wavelets. Adaptive wavelets are used by researchers to find a basis for measuring rates of decrease or increase in coefficients ([GRAP95]). DWT has many uses in computer and human vision analysis, FBI fingerprint compression, "denoising" noisy data, detecting self-similar behavior in a time-series of images, and even musical tone generation. In adaptive quantization with wavelets there are two popular methods: (1) fixed quantization for all coefficients in a given band using layered transmission of coefficients in binary (low) order arithmetic coding, and (2) using different quantizers and entropy coders for different regions of each subband. In particular a context-based adaptive arithmetic coder operating in the subband domain has been explored. Unlike the image domain the contexts are based upon past quantized data, not the original image data. Results from this novel algorithm indicate it performs as well as existing DWT methods without the use of trees ([CHRI96]).

DCT functions, on the other hand, do not possess localization of frequencies. The DCT takes advantage of the large amount of redundancy in images and the fact that there are constant-variance contours in the cosine transform over typical image sets. Usually a zonal coding strategy is sufficient to use these traits in order to assign a number of quantization levels based on the variance of the coefficient ([EMBR91]). The JPEG algorithm is significantly based on DCT, and unlike DWT it has been able to be implemented in both hardware and software, yielding compression ratios of over 50:1. DCT is also a fairly simple algorithm to program since it is based on only two dimensional FFT cosine functions (see Appendix A, Figure 6). So although DWT uses a variable kernel (window) size and infinite sets of possible short high-frequency and long low-frequency basis functions (some even having fractal structure), DCT will be used for experimentation in this paper. It should be noted that there are Wavelab and Matlab utilities available from Stanford University, but the programming presented in this paper is predominantly original code designed by the author. This was necessary from both a learning perspective and because it provided the flexibility of revising the code in fine detail to suit the project.

However, the programming was noticeably patterned after formal types of signal processing algorithms (e.g. edge-detection) as well as the DCT and bit-packing portions of the JPEG image compression algorithm.

4.2 Previous Work

Moderate amounts of work have been done in research to design image-adaptive quantization tables, especially in the field of medical imaging where image quality is paramount (see Appendix A, Figures 10 and 11). In an article by Good, JPEG is said to have difficulty in being applied to radiographic images because the standard does not specify a certain quantization table. This quantization table is a major part of what determines the compression ratio and quality degradation of compressed images. Usually psychovisual considerations are used in this field. However, these methods are dependent upon display characteristics and viewing conditions, so these considerations may not be optimal. This article reveals two methods for improving the current JPEG algorithm: (1) removing frequency coefficients where many bits are expected to be saved, and (2) preprocessing the image with a non-linear filter used to make the image more compressible. Optimally, the results should be robust enough so that they are qualitatively independent of a specific image or quantization table ([GOOD92]). That is, the quantization table should have the ability to be used on different images of the same type such as chest or cervical radiographic images ([BERM93]). The author's conclusions were that issues related to the use of the JPEG standard should not be confined exclusively to quantization tables. Subjective differences are more important on soft displays than on film, and even the MSE measurement does not always correlate greatly with the visual evaluation of images by human observers ([GOOD92]). Other methods may be better suited to imitate such subjective classifications.

Another such paper, written by Fung and Parker, presents an algorithm to generate a quantization table that is optimized for a given image and distortion. The most popular current practice in JPEG is to scale the table according to a scaling curve, but this method is not optimal because the bit rate and distortion rate are not both minimized. In theory, the optimal quantization table can be determined with an exhaustive search over the set of possible quantization tables. With values falling between 1 and 255, there is no single uniform quantizer for each DCT coefficient. Furthermore there are 255^{64} possible quantization tables, so an exhaustive

search over all possible values is impractical. The algorithm presented by these authors starts with an initial quantization table of very coarse quantizers, then the step sizes of the quantizers are decreased one at a time until a given bit rate is reached. At each iteration the element of the quantization table to be updated is varied depending on whether a coarser or finer quantization table is needed to achieve the target distortion value. The resulting algorithm has a poor computational complexity until an entropy estimator is used to estimate the bit rate and further reduce the complexity. Convergence is achieved when two points oscillate from one to another as downslope and upslope become about the same. A mean-squared quantization error (MSQE) is used for a distortion measure because it swiftly calculates it from unquantized and dequantized DCT coefficients and because MSE is more computationally expensive. Also, it can be weighed by the Human Visual System (HVS) response function without complication so that the weighted MSQE corresponds to perceptual quality. The bit rate estimate was measured using the entropy of quantized values without actually going through the entropy coding. The results showed that the optimized image-adaptive quantization table yields a 14% to 20% reduction in bit rate compared to algorithms which use standard scaled quantization tables. Once an optimized table is made for a given image, it can then be applied to similar images with only a small increase in bit rate. That is, the algorithm needs to be run only once for a given set of similar images ([FUNG95]).

There have been very few attempts at using GA-type heuristics to solve the problem of adaptive quantization for image compression. But in an article by Kil and Oh, they outline vector quantization based on a genetic algorithm. Vector quantization (VQ), although not specifically related to image coding, is the mapping of a sequence of continuous or discrete vectors (signals) into a digital sequence. The goal of VQ is to minimize communication channel capacity or storage needs while maintaining the fidelity of the data. In this particular example, the authors try to find the optimal centroids which minimize the chosen cost function by iterating through the selection, crossover, mutation, and evaluation operations associated with a typical GA. LBG (Linde, Buzo, Gray) and LVQ (learning vector quantization) are algorithms used as update rules for the centroids, which are represented as 64 bit (8x8) chromosomes. Each centroid (i.e. vector) is initially evaluated according to the error function, then selection is performed based on a certain probability. Next, a recombination called a crossover occurs at a random position in the string. Subsequently each bit of the string undergoes a mutation

based on a certain probability. Finally, centroids are "fine-tuned" using an LVQ update rule. The ultimate simulation contained 3 sets of data generated from Gaussian, Laplacian, and Gamma probabilities. A total of 10000 training samples were created over 100 generations, and a population size of 20 chromosomes was used. The crossover and mutation probabilities were 0.25 and 0.01, respectively. LVQ was applied to chromosomes with or without the GA applied. Results indicated that it is very effective to quantize image data compared to LVQ alone because the computational complexity is reduced ([KILO96]).

In other articles it has been shown that the Kohonen self-organizing map (SOM), which is strongly related to LVQ, can be used in several different ways for image compression. One such article by Yapo and Embrechts discusses the use of a Kohonen self-organizing feature map (SOFM) to make a codebook for coding an image upon which the SOFM was trained. This type of neural vector quantization has three phases including the design of the codebook using best matched blocks created by the SOFM, the compression of the codebook using a lossless method, and the coding of the image by finding the closest codebook block for each image block in $O(N)$ or $O(logN)$ time. The image reconstruction is done in reverse order, and mean-squared error (MSE) and signal-to-noise ratio (SNR) are used for comparisons. The results of applying the SOFM to VQ are promising because the codebooks produced can accurately quantize images for which they are trained and the resulting codebooks can be compressed. However, more research is still required to determine if this use of a SOFM is worthwhile ([YAPO96]).

An article by Xuan and Adah builds on the ideas of Yapo and Embrechts by introducing a Learning Tree Structured Vector Quantization (LTSVQ) codebook design. The SOFM proved to have good performance compared to the LBG (Linde, Buzo, Gray) algorithm, but it is limited to small vector dimensions and codebook sizes for most practical problems. The LTSVQ codebook design scheme uses a competitive learning (CL) algorithm to make an initial codebook. Then it partitions the training set into subsets which are used to design a new codebook. The results showed that the algorithm had good performance with low computational complexity.

In another related article by Anguita, Passaggio, and Zunino a SOM-based interpolation scheme is created for the purposes of image compression. VQ uses Euclidean metrics to associate a "prototype" with each data

sample, and its main advantage is that prototypes are expressed in the same form and domain as the processed data. However, VQ does attain high compression ratios at the expense of lower visual quality. The "multi-best" algorithm presented uses the interpolation process instead of VQ schemes to correctly reconstruct the unknown patterns not in the codebook. Thus, the codebook is more well-defined and yields a better reconstruction of details. The system is also sensitive to the training set adopted, and it uses more than one reference vector to code an image block. The algorithm is applied to a large set of natural images allowing reliable evaluations. The experimentation, using a Kohonen SOM with 256 neurons, showed that associated generalization properties greatly compensates for less compression. The details are reconstructed better and the edges are more accurate. These results were reinforced using analytical (i.e. MSE) and qualitative measures. Compared to VQ and other DCT-based methods, this technique is suitable for very high compression ratios ([ANGU95]).

All of the preceding resources were used as inspiration for the new method for solving the problem of making a set of edge-adaptive quantization tables presented in this paper. Many different evaluation schemes utilizing bits per pixel (BPP) and peak signal-to-noise ratio (PSNR) are examined. The details of the GA, TS, and image block classification algorithms used still require some explaining.

4.3 Theoretical and Conceptual Development

Several researchers including David Goldberg, Philip Segrest, Lawrence Davis, Jose Principe, Michael Vose, Gunar Liepins, and Allen Nix have created mathematical models of simple genetic algorithms which capture all associated details in the mathematical operators utilized. Mitchell outlines the formal model developed by Vose and Liepins concisely. First, start with a simple random population of binary strings having length l. Then calculate the fitness $f(x)$ of each string x in the population. Next, using replacement, select two parents from the current population with a probability that is proportional to each string's relative fitness in the overall population. Then crossover the two parents at a single random point with a certain probability to form two offspring. If no crossover occurs the offspring are exact copies of the parents. Finally, mutate each bit in the chosen offspring(s) with another probability, and place one (or both) in the new population. Normally only one of these children survive and the other is discarded for the next generation, but simple genetic algorithms allow

both to live ([MITC97]). This selection, crossover, and mutation process continues until the globally optimum solution is achieved or the upper bound on iterations is reached.

Although there is no rigorous answer to when a genetic algorithm (GA) yields a good answer to a given problem, many researchers share the intuition that a GA has a good if not better chance at quickly finding a sufficient solution than other weaker methods if the environment is right. That is, if the search space is large, not well understood, not perfectly smooth, has a single smooth hill (unimodal), has a noisy fitness function, and the task does not need a globally optimum solution, then a GA may be a good algorithm to use (see Appendix A. Figure 8). This is especially the case when alternative methods have no domain-specific knowledge in their search procedure. The coding of candidate solutions is generally central to the success of a GA ([MITC97]).

Another heuristic called tabu search (TS) was developed by Fred Glover as a strategy to overcome the problem of finding only local optima as a solution for mainly combinatorial optimization problems ([HOUC96]). Instead of only accepting solutions leading to better cost values. this search method examines the whole neighborhood of a current solution to take the best one as the next move even if the cost value gets worse. Optimization is usually performed by an iterative "greedy" component as a modified local search to bias the search toward points with low function values, while using prohibition strategies to avoid cycles ([TOTE95]). It is different from the traditional hill climbing local search techniques in the sense that it does not become trapped at locally optimal solutions. That is. it allows moves out of a current solution which makes the objective function worse in hopes that it will eventually find a better solution ([HURL97]).

Generally, a TS requires certain specifications in order for it to work properly. First, it needs a *configuration* which is the usage of a solution as an assignment of values to variables in an admissible search space (sometimes called a "search trajectory"). Second, it needs a *move* which is the process of generating, from a local neighborhood, a feasible solution to the combinatorial problem that is related to the current solution. Third, it requires a *set of candidate moves* out of the current configuration based on a cost function. If this set is too large, a subset of this set will suffice. Fourth, it should have specific *tabu restrictions* as conditions imposed on moves which make some forbidden for a certain "prohibition period." This period can be represented by a tabu size of

iterations, and it may be dynamically adjusted to handle cycles. This includes a tabu list of a certain size that records forbidden moves. Fifth, it should also have *aspirational criteria* as rules which regulate diversification and allow the override of a given tabu restriction (see Appendix A, Figure 8).

Tabu search can been used for optimization in manufacturing, planning, scheduling, telecommunications, structures and graphs, constraint satisfaction, financial analysis, routing and network design, transportation, neural networks, and parallel computing among others ([GLOV97]). Although there have been no citings of TS being used for image processing, this thesis attempts to use it as an alternative to the GA method previously described.

For classification purposes, a simple Kohonen SOM operates in such a way that the resulting neighborhood is set so that only one cluster updates its weights at each step, and only a certain low number of clusters are selected (see Appendix A, Figure 7). The learning rate is slowly (and linearly) decreased as a function of time while the radius of the neighborhood around a cluster unit decreases as the clustering process progresses. The formation of a map occurs in two phases by initially forming the correct order and then having the final convergence. The second phase takes longer than the first as the learning rate drops, and the initial weights may be random. Some applications of the Kohonen SOM include computer-generated music, the traveling salesman problem, and character recognition among many others like the ones mentioned in the articles previously presented.

Other classification strategies specific to edge-detected images include skeletonization and perimeterization. Skeletonization is the process of stripping away the exterior pixels of an edge pattern without effecting the pattern's general shape ([AZAR97]). Perimeterization achieves a similar result by using the connected neighborhood of adjacent pixels for each edge pixel in order to strip away the interior of edges found by the edge-detection algorithm (see Appendix A, Figure 7). A generalization of this perimeter determination, based on the relative percentage of edge area found per image block, is used as a simple substitute for the Kohonen SOM classification scheme.

4.4 Glossary

BPP – bits per pixel

DCT – discrete cosine transform

GA – genetic algorithm

JPEG – Joint Photographers Experts Group

MSE – mean-squared error

RMS – root mean-squared error

SOM – self-organizing map

PSNR – peak signal-to-noise ratio

TS – tabu search

5 Implementation

The general statement of the problem given earlier only gives a cursory view of what is to be done for this thesis. The following subsections elaborate more on what exactly the program does (see Appendix B, Figure 1). The user and technical documentation of Appendix C gives additional information about the development of the program.

5.1 Functional Description

The *main* function takes as arguments from the command line those parameters which are required by the *gacompress, tscompress,* and *classifyimage* routines. The command line is fairly rigid in format with arguments at specific locations, and there are many specialized functions for this application. The GA and TS algorithms are confined to the *gacompress* and *tscompress* functions, respectively. Each of these functions receives as parameters the number of bits per pixel, kernel size, image rows, image columns, number of quantization matrices used, maximum iterations allowed, input filename, output filename, and the compressed image filename. However,

the GA has tournament size, population size, crossover type, crossover probability, and mutation probability as additional inputs whereas the TS only has tabu size as the extra input. These two functions are subdivided into routines defining the different main operations that each perform initially and iteratively.

Before entering either of these major routines the image must be classified into a given number of quantization matrix types by calling the *classifyimage* function. This function takes as arguments the number of bits per pixel, kernel size, image rows, image columns, number of quantization matrices used, input file, output file, and the Kohonen SOM inputs consisting of maximum clusters, initial radius, maximum iterations, threshold value, learning rate, and adjustment rate. Since the Kohonen network has been disabled only the maximum clusters parameter is used for the simple relative edginess form of classification mentioned earlier. These classification parameters are fed into the *claskohonen* routine as labeling of each block of image data proceeds.

The *compressimage* and *expandimage* routines each take as parameters the number of bits per pixel, kernel size, image rows, image columns, number of quantization matrices, input file, and output file. These functions control the compression and expansion of the images, as dictated by the *gacompress* and *tscompress* routines by iteratively calling packing and DCT functions for each image block. The associated *packrec* function takes as input the number of bits per pixel, kernel size, quantization matrix index, and the packed and unpacked buffers. This function uses *packbits* for packing and unpacking bits within a block (record), using the storage bits based on the assigned quantization table instead of the kernel size. Scaling the DC coefficient, as well as applying the zig-zag pattern of block coding, is controlled in the *packrec* function. Note that the quantization matrix values for each kernel member of each image block is used to multiply the corresponding packed block kernel member upon retrieval and divide the unpacked block kernel member upon storage (see Appendix N for DCT dissection).

The global matrices used by the program must be created by separate functions. The *makerefbits* function simply stores the possible powers of 2 being used, and it only takes the number of bits per pixel as a parameter. The *makestorematrix* routine makes the storage matrix (how many bits to store in each block) based on the parameter input of bits per pixel, kernel size, and the specific quantization table index. The *makecosinematrices* function makes both forward and inverse DCT matrix kernels by taking the kernel size as input.

All associated matrix operations have their own separate functions within the program. The *matrixinsert* function allows insertion and extraction of short-integer image block data to and from the image data matrix, and it takes as parameters the number of bits per pixel, kernel size, image row offset, image column offset, image rows, image columns, image data matrix, and the block data matrix. The *matrixconvert* routine converts a floating-point matrix to a short-integer matrix (and vice versa), and it takes as input the bits per pixel, kernel rows, kernel columns, the floating-point matrix, and the short-integer matrix. The *matrixmultiply* function simply multiplies two floating-point matrices, and it takes as parameters the rows and columns of each image block as well as the input and output buffers. The *matrixtranspose* function transposes a given matrix, based on its rows and columns, using input and output buffers. The *matrixshow* routine is only used for matrix display debugging, and it takes as input the rows, columns, and buffer of the image block being handled.

The function *edgedetect* performs edge-detection on image data to produce an edge-only representation, and it takes as input the bits per pixel, image rows, image columns, image data buffer, edge data buffer, and a threshold value for determining what resulting code values correspond to an edge. A Gaussian blur is done on 9 byte (3x3) blocks of image data followed by two dimensional convolutions with Sobel or Laplacian filter kernels, then the threshold is applied. The associated *convolve2d* function takes as input the kernel rows, kernel columns, filter matrix, input matrix block, and output matrix block.

There are only two functions responsible for file input and output. The *accessimage* function takes a flag for compressed or uncompressed data, bits per pixel, kernel size, image rows, image columns, filename, and image data matrix in order to perform the appropriate read or write operation. It uses the *testaccessbyte* routine to load or unload one byte at a time associated with the image data matrix, and it takes as parameters a flag for writing or reading data, the byte to access, the current bit value to access, the current bit count, and the pointer to the file structure being used for access.

There are also various miscellaneous and important supplementary functions. The *dct2d* function uses the *matrixmultiply* routine to perform the discrete cosine transform on an image block (i.e. in two dimensions), and it takes as arguments the number of bits per pixel, block rows, block columns, and the short-integer matrix

buffer. The *evalimages* routine evaluates an image to find BPP, MSE, RMS, PSNR, and the cost ratio (RAT). Its arguments are the output values for these calculations as well as kernel size, image rows, image columns, number of quantization matrices, test matrix buffer, and reference matrix buffer. The *getrandomint* function gets a random integer within the range entered as parameters. The *getrandomflt* function gets a random floating-point number between 0 and 1. In addition, the *randomize* routine should always proceed either of these two functions in order to initialize the random number generator based on a random number seed.

5.2 System Design

There are 8 different modules for this application. The *quanmain* module contains the *main* routine as well as a subordinate *testtest* function. Since classification comes first, the *quanclas* module containing the *classifyimage* and *claskohonen* functions is next in the hierarchy. The affiliated *quanedge* module contains the *edgedetect* and *convolve2d* functions that are used for edge-detection. The *gacompress* and *tscompress* routines, as well as their component functions, are located in the *quanadap* module. Next is the *quanpack* module containing the *compressimage* and *expandimage* functions in addition to the *packrec* and *packbits* helper routines. The order of the other modules is somewhat hazy since their usage is intermixed. The *quanmake* module contains the *makerefbits, makestorematrix,* and *makecosinematrices* functions mentioned earlier. The *quanmatr* module contains all practical matrix operations including the *matrixinsert, matrixconvert, matrixmultiply, matrixtranspose,* and *matrixshow* functions. Finally, the *quanmisc* module contains the *accessimage, testaccessbyte, dct2d, evalimages, getrandomint, getrandomflt,* and *randomize* routines responsible for the remainder of the mathematics and file transactions.

The flow of data between these functions and modules is shown in Figure 1 of Appendix B. The equipment configuration consists of any SUN Workstation running under version 2.6 of Solaris (i.e. in a UNIX environment). In addition, an implementation tool (e.g. osiris) is required for psychovisual testing in order to supplement the objective evaluation using the cost function given in Figure 9 of Appendix A.

5.3 Decisions and Tradeoffs

This *quanmain* program is not capable of processing anything other than 8 bit images that are 256 rows by 256 columns in dimension. Also, for quantization the program only accepts an 8 byte kernel size and 1, 2, 4, 8, or 16 for the number of quantization matrices. All other directions given in Figure 2 of Appendix B should be followed explicitly. These limitations made the program easier to code such that more focus was able to be placed on the task of finding an optimum set of quantization tables for an image.

The entire project could have been better outlined using an object-oriented (OO) software engineering approach, ultimately resulting in the code being written in an OO language (e.g. C++). However, the functional design of the code, along with the descriptive comments, allows for an easy understanding about how all of the components of the thesis project fit together.

5.4 Experimentation

As stated in the Technical Documentation of Appendix C, the code of each function in every module was tested independently. However, the functions in the *quanadap* module were tested in conjunction with all other code to prove their correctness. A total of 11 raw images, 5 similar abdominal images and 6 others of varying edginess, were used for experimentation. Some of the image types were radiographic in origin. The purpose of all subsequent analysis is to compare recovered images to the originals and their lowest-compressed forms. Also, the objective and subjective results produced for all variations of classification and optimization were systematically contrasted (see Appendix B, Figure 8). Note that the lowest-compressed images were compressed and expanded with a quantization matrix consisting of only 1s, using only the code written for this thesis. All calculations were verified using a calculator.

6 Analysis

The reference images for all original, lowest-compressed, Sobel edge-detected, and Laplacian edge-detected images are given in Appendices D through G, respectively. All other resulting expanded images, analysis, results, and conclusions are detailed in Appendices H through M. The following subsections are based upon these appendices.

6.1 Results and Conclusions

The results using GA and TS optimization methods for finding the best set of quantization matrices for an image are given in Figures 1 through 16 of Appendix M. The analysis of these results yields many interesting observations. First, the edge-transitions produced from the perimeterization method used for block classification, gives the best results overall. Second, only sets of 1 or 2 quantization matrices used for both GA and TS methods yield the best objective results, employing the basic cost function without weights. Third, a set of quantization matrices designed for an abdominal image works well on other similar abdominal images. Fourth, both subjectively and objectively the GA method appears to outperform the TS method in just about every respect. Many other less significant results pertaining to how each optimization method performed can be viewed in Figure 15 of Appendix M. The computational complexity of each algorithm is detailed in Figure 16 of the same appendix. The GA ran in a faster $O(N)$ time that of the TS algorithm due to the fact that tabu search required the examination of each possible kernel move in every iteration (where N is the number of pixels in an image).

In looking at all the resulting data more firm conclusions can be made. Subjectively, both optimization methods appear to produce recovered images which are less detailed and more blurry than both the original and lowest-compressed set of images. However, the recovered images did appear to show less blockiness around the edges in all of the test images, and that was one of the aims of this thesis. Perimeterization used with the Laplacian operator appears to provide the best means for classification of the image blocks. Although the GA outperforms TS in both cost and computational complexity (time duration), both methods worked well

after applying a quantization matrix of an image to images of the same type. The GA performed best using a two-point crossover, a crossover rate of about 0.90, a mutation rate of about 0.06, a tournament size of around 30, and a population size of around 30. The TS method performed best using a tabu size of around 70. All of these conclusions take into account objective and subjective analysis (see Appendix M, Figure 17).

It is too difficult to tell what amount of quantization matrices performs best using the weightless cost function given in Figure 9 of Appendix A. However, preliminary results show that an optimum number of quantization matrices can be found for any image using the weighted cost function. This amount appears to vary based upon the relative edginess of certain images (i.e. more edginess means more quantization matrices for classification). Results also indicate that raising the PSNR weight from 1 to 32 (by powers of 2) has a negative affect on the BPP calculation only, while raising the BPP weight in a similar manner has a negative affect on the PSNR calculation only. Objective and subjective results were also used in this analysis.

6.2 Problems and Discrepancies

Many problems occurred during the implementation of this thesis, but all of them were able to be overcome. One of the major hurdles was getting the initial compression and expansion of an image to work. To beat this problem the DC (upper left) term of each image block had to be scaled and descaled when packing and unpacking, respectively. Another problem that was able to be solved was finding an appropriate weighted cost function for allowing better image quality or better compression based upon user preference. Over several weeks many different variations were tested, but in the end the alternate cost function given in Figure 9 of Appendix A was chosen for extra testing.

There were some peculiarities that transpired which were not expected in the results. It was intended that more quantization tables, regardless of the degree of edginess of a given image, would always be beneficial. This did not turn out to be the case after analyzing the data. Also, the Sobel edge-detection operator did not appear to be good for classification, but in some cases it did perform better on the same images whose blocks were classified with the Laplacian operator.

6.3 Lessons Learned

As stated previously I would rather have attempted this thesis from more of an object-oriented standpoint. This would have put an envelope of inheritance and expandability around the project such that others could more easily understand the components and add more where necessary. However, since DCT is apparently being replaced by DWT for the future JPEG standard, the likelihood of someone adding to this exact thesis project is fairly remote. It should be stated, though, that the techniques used within this thesis may very well have applications to the different type of quantization contained within DWT.

In reference to the articles inspected in this thesis, Good said that psychovisual experiments are sometimes dependent upon suboptimal display characteristics. The weighted cost function used for alternate studies in this thesis proved to be a fairly good substitute (although not a replacement) for such experiments. Quantization tables were able to be successfully applied to images of similar types, just as it was proclaimed in the article by Berman, Long, and Pillemer. Although estimations were not used for BPP and PSNR calculations, the weighted cost function and initial random coarse quantization matrices were similar to those mentioned in the article by Fung and Parker. It should be noted that although using estimations would have reduced the running time of the program, the computational complexity of the algorithms used would still have been asymptotically the same. The balance of image quality and compressibility for transmission or storage was implemented in a comparable manner to the methods described in the article by Kil and Oh.

Possible future enhancements, such as adding the capability for applying the algorithms to larger and denser images, can be made to the code. Also, although it is not recommended, the kernel size could be made more variable. Skeletonization may be used for classification instead of the modified perimeterization technique, and other neural networks can be examined to find out their ability to help classify the blocks of an image. Moreover, lossless coding algorithms may be used to increase the compression rate even further. Of course, simply modifying the quantization tables is not a panacea for any type of image compression. Other factors, including those that may have nothing to do with edge-detection, should be considered when trying to create an overall better image compression algorithm.

Although I had never done investigations or projects related to image compression before doing this thesis, I believe I now have the capability of applying my knowledge of image compression optimization to other compression strategies. This has truly been an enlightening experience for me and I am grateful for all the of people who have helped me in this attempt.

7 Bibliography

URL REFERENCES:

Azar. Danielle. jeff.cs.mcgill.ca/~godfried/teaching/projects97/azar/skeleton.html, 1997.

Compression Overview. www-ise.stanford.edu/class/ee392c/demos/bax_and_vitus/node6.html, 1997.

Data Compression Reference. www.rasip.fer.hr/research/compress/algorithms/fund/ac/index.html, 1997.

Edge-detection. www.cclabs.missouri.edu/~c655467/CECS_365/Object_Img/Lapla_Img/part_3.html, 1997.

Hurley. S. www.cs.cf.ac.uk/User/Steve.Hurley/Ra/year2/node14.html, 1997.

Measures of Image Quality. seurat.uwaterloo.ca/~tveldhui/MAScThesis/node18.html, 1997.

Smith, Steven W. www.dspguide.com/datacomp.htm, 1996.

Totem (The Reactive Tabu Search). infn.science.unitn.it/totem/totem95/node2.html, 1995.

BOOK REFERENCES:

Althoen, Steven C. and Bumcrot, Robert J. Matrix Methods in Finite Mathematics. New York, NY. W. W. Norton and Company, 1976.

Embree, Paul M. and Kimble, Bruce. C Language Algorithms for Digital Signal Processing. Englewood Cliffs, NJ, Prentice-Hall Inc., 1991.

Fausett, Laurene V. Fundamentals of Neural Networks. Englewood Cliffs, NJ, Prentice-Hall Inc., 1994.

Glover, Fred W and Laguna, Manuel. Tabu Search. Norwell, MA. Kluwer Academic Publishers, 1997.

Mitchell, Melanie. An Introduction to Genetic Algorithms. Cambridge, MA. The MIT Press, 1997.

Russ, John C. The Image Processing Handbook. Boca Raton, FL, CRC Press, 1992.

PAPER REFERENCES:

Anguita, Davide, Passaggio, Filippo, and Zunino, Rodolfo. SOM-Based Interpolation for Image Compression. World Congress on Neural Networks, Washington, DC, Vol. 1, 1995.

Berman, L. E., Long, R., and Pillemer, S. R. Effects of Quantization Table Manipulation on JPEG Compression of Cervical Radiographs. Society for Information Display International Symposium, Volume 24, 1993.

Christos, Chrysafis and Ortega, Antonio. Context-based Adaptive Image Coding. Integrated Media Systems Center, USC, Los Angeles, CA, 1996.

Distasi, Riccardo, Nappi, Michele, and Vitulano, Sergio. A B-Tree Based Recursive Technique for Image Coding. 13th International Conference on Pattern Recognition, Vienna, Austria, Volume 2, 1996.

Fung, Hei Tao and Parker, Kevin J. Design of Image-adaptive Quantization Tables for JPEG. Journal of Electronic Imaging, April, 1995.

Good, Walter F Quantization Technique for the Compression of Chest Images by JPEG Algorithms. SPIE Volume 1652 Medical Imaging VI: Image Processing, 1992.

Graps, Amara. An Introduction to Wavelets. IEEE Computational Science and Engineering, Summer, 1995.

Houck, C. R., Joines, J. A., and Gray, M. G. Characterizing search spaces for Tabu Search. Dept. of Industrial Engineering, North Carolina State University, Raleigh, NC, 1996.

Jiang, J. Algorithm Design of an Image Compression Neural Network. World Congress on Neural Networks, Washington, DC, Volume 1, 1995.

Kil, Rhee M. and Oh, Young-in. Vector Quantization Based on Genetic Algorithm. World Congress on Neural Networks, Washington, DC, Volume 1, 1995.

Ramos, Marcia G. and Hemami, Sheila S. Edge-adaptive JPEG Image Compression. School of Electrical Engineering, Cornell University, Ithaca, NY. 1996.

Xuan, Kianhua and Adah, Tulay. Learning Tree-Structured Vector Quantization for Image Compression. World Congress on Neural Networks, Washington, DC, Vol. 1, 1995.

Yapo, Theodore C. and Embrechts, Mark J. Neural Vector Quantization for Image Compression. Rensselaer Polytechnic Institute, Troy, NY. 1996.

8 Appendices (all subsequent pages)

Masters Thesis (Winter 1998/1999)

APPENDIX A – Background Information

Michael S. Champion (e-mail: msc0686@cs.rit.edu)

January 28, 1999

Genetic Algorithm & Tabu Search Approaches to Adaptive Quantization in DCT Image Compression

(by Michael Champion for MS Thesis in Computer Science, Winter 98/99)

Today there are several formal and experimental methods for image compression, some of which have grown to be incorporated into the Joint Photographers Experts Group (JPEG) standard. Of course, many of the associated algorithms are still used only for experimentation mainly due to various performance issues. Lack of speed while compressing or expanding an image as well as poor image quality after expansion are a couple of the most popular reasons for skepticism about a particular compression algorithm.

This paper discusses current methods used for image compression. It also gives a detailed explanation of the discrete cosine transform (DCT), used by JPEG, and the efforts that have recently been made to optimize related algorithms. Some interesting articles regarding possible compression enhancements will be noted, and in association with these methods a new implementation of a JPEG-like image coding algorithm will be outlined. This new technique involves adapting between one & sixteen quantization tables for a specific image using both genetic algorithm (GA) and tabu search (TS) approaches. First, a simple algorithm will be used to classify the edge-detected blocks of a given image. A complex Kohonen self-organizing map algorithm is attempted, but the results using this algorithm appeared to be worse due to lack of convergence upon a trusted set of weights. Next, the GA and TS algorithms will be used to iteratively seek out the optimum quantization matrices. The selected cost function minimizes bits per pixel while maximizing the peak signal-to-noise ratio after the compression and expansion of the original image.

Figure 1: Thesis Introduction

LOSSLESS CODING METHODS:

Lossless coding methods are able to restore a compressed image that is exactly identical to the original image.

Lossless coding techniques include optimal entropy coding, Huffman coding, Lempel-Ziv coding, adjacent pixel pair coding, predictive pixel coding, lossless pyramid coding, and differential pulse code modulation (DPCM). Optimal entropy coding works in such a way that the length of each codeword is proportional to the frequency used, and it has achieved 7% compression at 7.39 bits per pixel (bpp). Huffman coding is an example of this type of uniquely decodable variable length coding method in which the code is made by examining the probabilities of source symbols and assigning codewords to minimize the average code length for instantaneous recognition. The Lempel-Ziv method which usually uses Huffman coding operates on a sliding window of data which is parsed until a dictionary is created, and it is able to achieve 18% compression with 6.54 bpp. Lempel-Ziv is a member of a set of many dictionary coding methods in which the dictionary does not always have to be sent with the data. Adjacent pixel pair coding exploits the correlation of neighboring pixels by selecting a code in which sequential pairs of pixels can be represented by a symbol, and it has achieved 25% compression with 5.99 bpp. Predictive pixel coding chooses a pixel value based on its neighbors, and it has achieved 39% compression with 4.85 bpp. Lossless pyramid coding progressively filters and subsamples the image into separate levels of decreasing size until a small dataset of low-frequency components remains and is encoded, and it has achieved 15% compression at 6.76 bpp. DPCM, which is often combined with many lossless and lossy coding methods, works in such a way that codewords represent the differences between sample values ([COMP97]). The Graphical Interchange Format (GIF), Portable Network Graphic (PNG), and Tag Image File Format (TIFF) standards all utilize some form of lossless coding, usually coupled with means to support lossy compression methods as well. These formal designs minimally include header, palette, and data areas ([DATA97]).

Figure 2: Lossless Coding Background

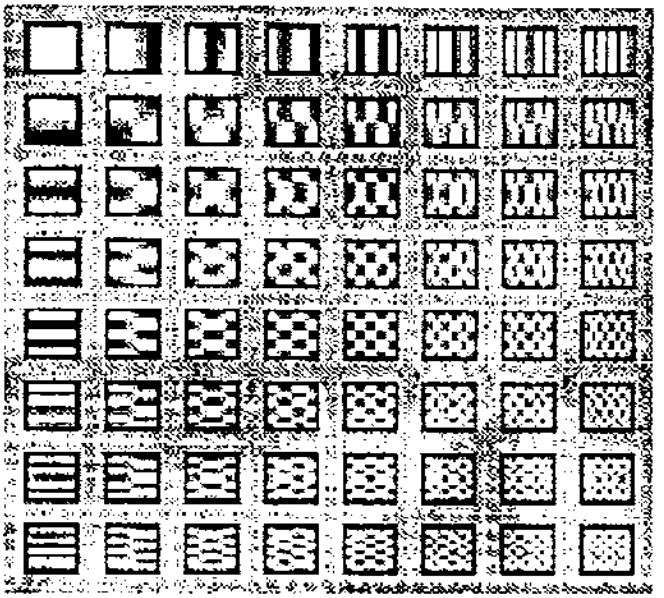
LOSSY CODING METHODS:

Lossy coding methods compromise quality for a lower transmission rate by quantizing in order to minimize irrelevant information.

Lossy coding techniques include the discrete cosine transform (DCT), discrete wavelet transform (DWT), arithmetic coding, pulse code modulation (PCM), run-length (chord) coding and Shannon-Fano coding. DCT divides an image into blocks which are transformed in such a way that output coefficients can be quantized according to a step size, and it has resulted in a signal-to-noise ratio (SNR) of 31.5dB at 1.33 bpp. DCT methods are quite often coupled with one or more lossless techniques like optimal entropy coding, pyramid coding, or DPCM. Adaptive DCT methods most often used with video image compression can reduce bitrates by a factor of two if an adaptive algorithm is used to optimize quantization and entropy coding separately for each block. DWT, often used to process x-ray and magnetic-resonance images, analyzes an image and converts it into a set of mathematical expressions that can be decoded by a receiver. Arithmetic coding works on the probability of source symbols and their encoding interval ranges which determine compression efficiency. It does have problems of real number underflow, overflow, and error sensitivity to single-bit corruption. However, it does have the flexibility of using static (fixed probability of symbols) or adaptive (dynamically estimated probability based on changing frequency) encoding. Pulse code modulation (PCM) involves quantization and digitalization of an analog signal which is sampled at regular intervals in order to divide it into segments with unique code values. The main goal of this method is to restore the signal so that it contains only components of the original signal above the threshold of human perception ([COMP97]). Run-length coding, considered to be lossless, simply compresses sequences of repeated signals in the original signal using the starting source symbol and the length of the run ([SMIT97]). Shannon-Fano coding uses a variable length of bits to encode source symbols according to their probabilities, and entropy is used to measure the error of the information source. Fractal image compression considers an image as copies of smaller parts of itself. Michael Barnsley of the Georgia Institute of Technology suggested that collections of affine transformations (skewing, stretching, rotating, scaling, translating, etc.) could lead to image compression. These collections, called Iterated Function Systems (IFS), are still undergoing research and a fractal image format (FIF) is still not standardized. However, the other lossy methods have been standardized in some shape or form in Microsoft Windows Bitmap (BMP) and Joint Photographers Group (JPEG) formats, among others ([DATA97]).

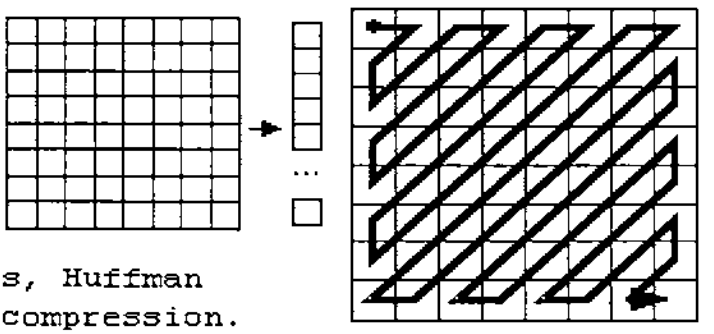
Figure 3: Lossy Coding Background

JPEG/DCT COMPRESSION OVERVIEW:



JPEG is a DCT-based form of image encoding in which DCT converts a 8x8 block of pixels into 64 coefficients representing the intensity of an 8x8 image block to be compressed.
High spacial frequency coef (lower right) occur less than low spacial frequency coef (upper left), so DCT coefficients can be stored with fewer bits than the original image block. This quantization yields a table for each compressed image.

Given the above situation, it is good to store coef in a zig-zag way to make a subsequent encoding method (Huffman or Arithmetic) better. Since many coef are zero & many run-sizes occur more often than others, Huffman coding is used for further compression.



The goal of this thesis is to enhance the process by allowing more than one quantization table to be applied per image. This idea is based on the premise that the Human Visual System uses greyscale edges to detect boundaries. The block-iness of DCT transformed images is more pronounced on edges, so GA & TS methods are used to find the quantization tables for the least to most edgy image blocks.

Figure 4: JPEG Overview

DWT VERSUS DCT:

There are 2 standard approaches for adaptive quantization in wavelets:
(1) fixed quantization for all coefficients in a given band and a layered transmission of coefficients using binary (low) order arithmetic coding
(2) using different quantizers and entropy coders for different regions of each subband (from [CHRI96])

$$W(x) = \sum_{k=-1}^{N-2} (-1)^k * c_{k+1} * \Phi * (2x + k)$$ (where Φ is the mother wavelet function)

c_k are wavelet coefficients which must satisfy linear and quadratic constraints like:

$$\sum_{k=0}^{N-1} c_k = 2 \quad and \quad \sum_{k=0}^{N-1} c_k * c_{k+2l} * \delta_{l,0}$$ (l is scaling int)

DCT, on the other hand, is one of the most common 2D transforms for image compression characterized by the following flow of data:

(transform compression)	*(recovery of image)*	
original image	*transform coded image*	
↓	↓	
forward transform	*expansion quantization*	
↓	↓	
coefficient selection	*coefficient zero fill*	
↓	↓	
quantization	*inverse transform*	
↓	↓	
transform coded image	*recovered image*	*(from [EMBR91])*

$$F(u,v) = \frac{1}{N*N} * \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(m,n) * \cos \frac{(2m+1)\pi * u}{2N} * \cos \frac{(2n+1)\pi * v}{2N}$$

$$\bar{f}(m,n) = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} c[u]*c[v]*F(u,v) * \cos \frac{(2m+1)\pi * u}{2N} * \cos \frac{(2n+1)\pi * v}{2N}$$

(where c[#] = 1 when # = 0; 2 otherwise)

Figure 5: DWT versus DCT

DERIVATION OF DCT FROM FOURIER TRANSFORM:

(CONTINUOUS FORWARD 1-D FOURIER)

$$F(u) = \int_{-\infty}^{+\infty} f(x) e^{-i*2*\pi*u*x} dx$$

(Euler's formula identity used below)

where $e^{-i*2*\pi*u*x} = \cos(2*\pi*u*x) - i*\sin(2*\pi*u*x)$

and $i=\sqrt{-1}$ (i.e. imaginery)

(sum of real & imaginery parts below)

$F(u) = R(u) + iI(u)$

(θ)

(CONTINUOUS REVERSE 1-D FOURIER)

$$f(x) = \int_{-\infty}^{+\infty} F(u) e^{i*2*\pi*u*x}$$

(DISCRETE FORWARD 1-D FOURIER)

$$F(u) = 1/N * \sum_{x=0}^{N-1} f(x) * e^{-i*2*\pi*u*x/N}$$

(DISCRETE REVERSE 1-D FOURIER)

$$f(x) = \sum_{u=0}^{N-1} F(u) * e^{i*2*\pi*u*x/N}$$

(DISCRETE FORWARD 2-D FOURIER)

$$F(u,v) = \frac{4c(u,v)}{N*N} * \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(m,n) * \cos\frac{(2m+1)\pi*u}{2N} * \cos\frac{(2n+1)\pi*v}{2N}$$

where N=width, u&v are in range 0 to N-1, and $c(u,v) = 1$

(DISCRETE COSINE TRANSFORM -- FORWARD)

$$DCT\{A\} = \frac{CAC^T}{N*N}$$

where (C) $c[u][n] = \cos\frac{(2n+1)u*\pi}{2N}$ (scaled by 1/N for calcs.)

(from [RUSS92] and [EMBR91])

Figure 6: Derivation of DCT

KOHONEN SOM ALGORITHM DESCRIPTION:

-- neighborhood of J (only certain low # clusters selected) is set so that only 1 cluster updates its weights at each step (i.e. R=0)

(1) Initialize weights w_{ij} & set neighborhood parameters, set learning rate parameters

(2) While stopping condition is false do the following:

(3) for each input vector (row) x, do the following:

(4) for each j, compute:

$$D(j) = \sum_{i}^{cols} (w_{ij} - x_i)^2$$

(5) find index J such that D(J) is a minimum

(6) for all units j within a specific neighborhood of J and for all i: $w_{ij}(new) = w_{ij}(old) + lr*[x_i - w_{ij}(old)]$

(7) update learning rate

(8) reduce radius of neighborhood at specific times

(9) test stopping condition

-- learning rate linearly decreased as a function of time

-- radius of neighborhood around cluster unit decreases as clustering process progresses

-- formation of map occurs in 2 phases: initial formation of correct order and the final convergence

-- may have random values for initial weights

EDGE-DETECTION ALGORITHM DESCRIPTION:

INPUT IMAGE → SMOOTHING FILTER → VERTICAL GRADIENT HORIZONTAL GRADIENT → GRADIENT COMBINING → APPLY THRESHOLD → BINARY EDGE-DETECTION

-- steps in edge detection are:

(1) convolve with Gaussian blur,

(2) find edges with vertical Sobel operator,

(3) find edges with horizontal Sobel operator,

(4) choose highest magnitude of Sobel operators,

(5) detect edges using threshold value (method for automatic thresholding possible)

Figure 7: SOM and Edge-detection Algorithms for Classification

GENETIC ALGORITHM DESCRIPTION:

Generally, a genetic algorithm starts with a simple random population of binary strings of given length (l). Then the fitness (f(x)) is calculated for each string (x) in the population. Next 2 parents are selected (with replacement) from the current population with a probability proportional to each string's relative fitness in the population. Now the 2 parents undergo crossover with a probability (p_c) to form 2 offspring. If no crossover occurs, the offspring are exact copies of the parents. Choose 1 offspring at random and discard the other (both survive in a simple GA). Mutate each bit in the selected offspring with probability (p_m) and place it in the new population. Iterate through the selection, crossover, and mutation operations until a new population is complete ([MITC97]).

TABU SEARCH DESCRIPTION:

A tabu search scheme can be defined as the evaluation of a criterion function for a current configuration. Then following a set of candidate moves the best non-tabu move is chosen to be in the next configuration. If all moves are tabu then the best one satisfying the aspiration criterion is chosen. If this cannot be accomplished, the best non-tabu move is picked for the current configuration. This is all repeated for a certain number of iterations. Upon termination, the best solution found is returned by the algorithm. When the tabu list is full, the first tabu element is freed and the process continues. Tabu moves are ideally all based on the short-term and long-term history of moves ([HURL97]).

Figure 8: GA and TS Algorithms for Optimization

RECOVERED VS. ORIGINAL IMAGE EVALUATION:

PEAK SIGNAL-TO-NOISE-RATIO (PSNR):

n = original bpp
S = max pixel value

$$PSNR = 10 * \log_{10}\left(\frac{2^{2n}}{MSE}\right) \text{ OR } PSNR = -10 * \log_{10}\left(\frac{MSE}{S^2}\right)$$

BITS PER PIXEL:

$$BPP = B / c_r = (N_1 N_2 \log_2 L + LM^4 B) / (N_1 N_2 M^2)$$

where B=bits in image, c_r=comp ratio

MEAN-SQUARED ERROR (MSE):

$$MSE = (1/(L*M)) * \sum_{i=0}^{L-1} \sum_{j=0}^{M-1} (x(i,j) - \bar{x}(i,j))^2$$

MEAN-SQUARED QUANTIZATION ERROR (MSQE):

$$MSQE = \sum_{n=0}^{N-1} \sum_{k=0}^{63} [z'_n(k) - z_n(k)]^2 \quad \text{(estimate of MSE)}$$

ENTROPY:

(actual)

$$E_f = \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} -[p(f(x,y)) \log_2 p(f(x,y))]$$

(estimate)

$$E = \sum_{s=0}^{M} \sum_{x=0}^{1} [-P(x)P(s|x)\log_2 P(s|x)]$$

where P(s|x) is probab of $s=S_n(k)$, P(x) is probab of $x=X_n(k)$

(from [JIAN93], [FUNG95], and [BERM93])

derived cost function (cf) = BPP / PSNR

alternate cost function = $cf + (BPP \cdot w_{bpp}) + (w_{psnr} / PSNR)$

Figure 9: Image Evaluation Methods

URL AND BOOK REFERENCES:

URL REFERENCES:

Edge-detection.
www.cclabs.missouri.edu/~c655467/CECS_365/Object_Img/Lapla_Img/part_3.html, 1997.

Data Compression Reference Center.
www.rasip.fer.hr/research/compress/algorithms/fund/ac/index.html, 1997.

Compression Overview.
www-ise.stanford.edu/class/ee392c/demos/bax_and_vitus/node6.html, 1997.

Totem (The Reactive Tabu Search).
infn.science.unitn.it/totem/totem95/node2.html, 1995.

Hurley, S.
www.cs.cf.ac.uk/User/Steve.Hurley/Ra/year2/node14.html, 1997.

Smith, Steven W.
www.dspguide.com/datacomp.htm, 1996.

BOOK REFERENCES:

Glover, Fred W. and Laguna, Manuel. Tabu Search. Norwell, MA, Kluwer Academic Publishers, 1997.

Mitchell, Melanie. An Introduction to Genetic Algorithms. Cambridge, MA, The MIT Press, 1997.

Fausett, Laurene V. Fundamentals of Neural Networks. Englewood Cliffs, NJ, Prentice-Hall Inc., 1994.

Russ, John C. The Image Processing Handbook. Boca Raton, FL, CRC Press, 1992.

Embree, Paul M. and Kimble, Bruce. C Language Algorithms for Digital Signal Processing. Englewood Cliffs, NJ, Prentice-Hall Inc., 1991.

Althoen, Steven C. and Bumcrot, Robert J. Matrix Methods in Finite Mathematics. New York, NY, W. W. Norton & Company, 1976.

Figure 10: References

PREVIOUS WORKS/ARTICLE REFERENCES:

*Good, Walter F. Quantization Technique for the Compression of Chest Images by JPEG Algorithms. SPIE Volume 1652 Medical Imaging VI: Image Processing, 1992.

*Berman, L. E., Long, R., and Pillemer, S. R. Effects of Quantization Table Manipulation on JPEG Compression of Cervical Radiographs. Society for Information Display International Symposium, Volume 24, 1993.

*Fung, Hei Tao and Parker, Kevin J. Design of Image-adaptive Quantization Tables for JPEG. Journal of Electronic Imaging, April, 1995.

*Kil, Rhee M. and Oh, Young-in. Vector Quantization Based on Genetic Algorithm. World Congress on Neural Networks, Washington, DC, Volume 1, 1995.

*Yapo, Theodore C. and Embrechts, Mark J. Neural Vector Quantization for Image Compression. Rensselaer Polytechnic Institute, Troy, NY, 1996.

*Xuan, Kianhua and Adah, Tulay. Learning Tree-Structured Vector Quantization for Image Compression. World Congress on Neural Networks, Washington, DC, Vol. 1, 1995.

*Anguita, Davide, Passaggio, Filippo, and Zunino, Rodolfo. SOM-Based Interpolation for Image Compression. World Congress on Neural Networks, Washington, DC, Vol. 1, 1995.

Ramos, Marcia G. and Hemami, Sheila S. Edge-adaptive JPEG Image Compression. School of Electrical Engineering, Cornell University, Ithaca, NY, 1996.

Distasi, Riccardo, Nappi, Michele, and Vitulano, Sergio. A B-Tree Based Recursive Technique for Image Coding. 13th International Conference on Pattern Recognition, Vienna, Austria, Volume 2, 1996.

Jiang, J. Algorithm Design of an Image Compression Neural Network. World Congress on Neural Networks, Washington, DC, Volume 1, 1995.

Houck, C. R., Joines, J. A., and Gray, M. G. Characterizing search spaces for Tabu Search. Dept. of Industrial Engineering, North Carolina State University, Raleigh, NC, 1996.

Graps, Amara. An Introduction to Wavelets. IEEE Computational Science and Engineering, Summer, 1995.

Christos, Chrysafis and Ortega, Antonio. Context-based Adaptive Image Coding. Integrated Media Systems Center, USC, Los Angeles, CA, 1996.

(these marked entries were focused on the most for comparison in the thesis)*

Figure 11: References (continued)

Masters Thesis (Winter 1998/1999)

APPENDIX B – Flowchart, Examples, and Codes Samples

Michael S. Champion (e-mail: msc0686@cs.rit.edu)

January 28, 1999

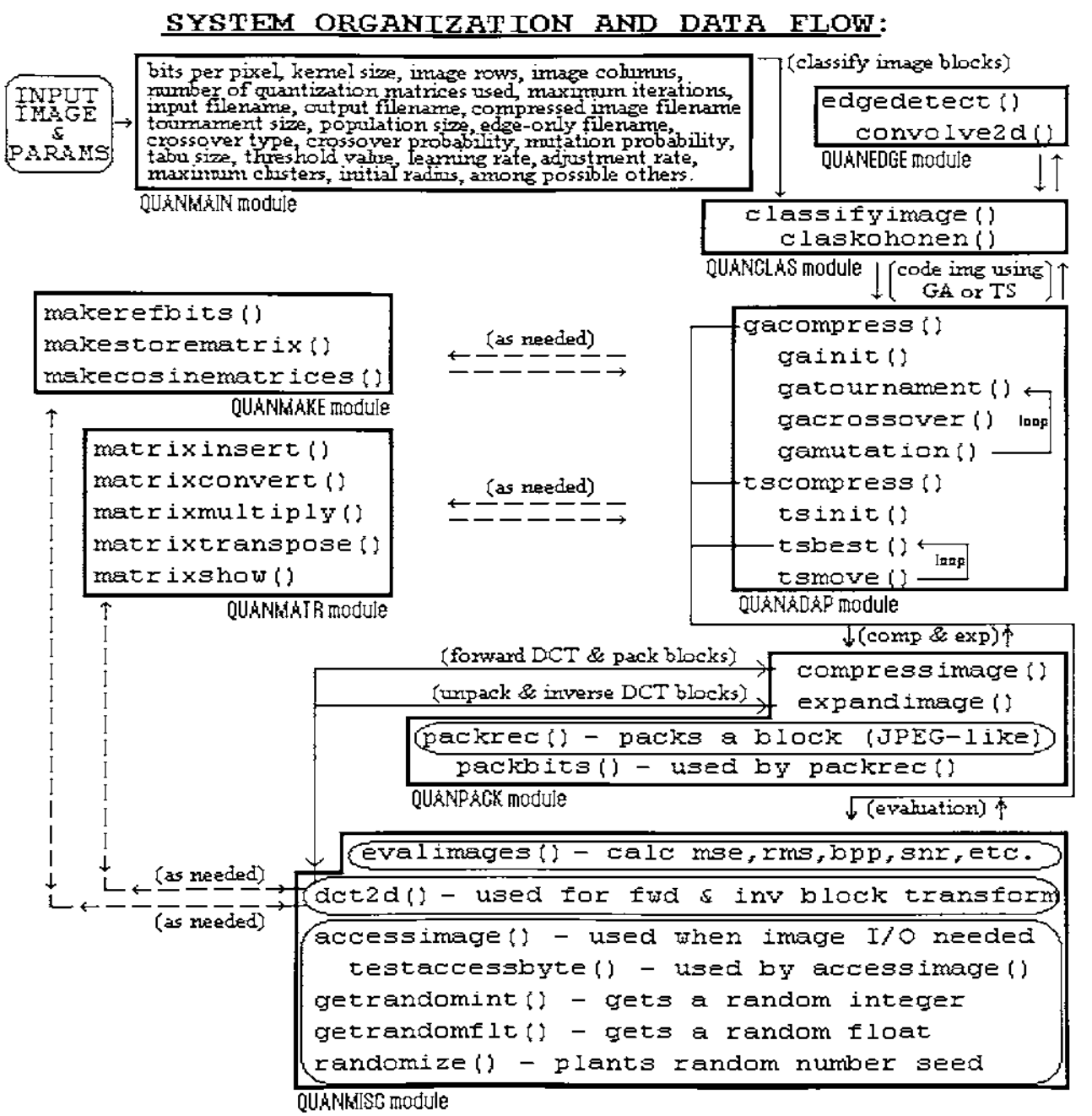


Figure 1: System Organization and Data Flow

INPUT FORMAT & EXAMPLE INPUT:

```
quanmain TY BU KS RO CO NQ MC IR KI TV LR AR IT TS [PS XO PC PM] OF EF CP XF
 where TY = TABU (for tabu search) or GENALG (for genetic algorithm) method
 where BU = number of bits per pixel in image
            (8 only,'-' if using edge transitions)
 where KS = kernel size for quantization
            (8 only,'-' if using Laplacian operator)
 where RO = number of rows of pixels in image (256 only)
 where CO = number of columns of pixels in image (256 only)
 where NQ = number of quantization matrices used in image (1,2,4,8,16)
 where MC = maximum clusters used for image classification (same as NQ)
 where TV = threshold value used for image classification (0-255,-1 = auto)
 where IR = initial radius used for image classification (0-MC)
 where KI = Kohonen iterations used for image classification
 where LR = learning rate used for image classification
            ('-' if cost bpp weight)
 where AR = learning rate adjustment for image classification
            ('-' if cost snr weight)
 where IT = iterations for tabu search or genetic algorithm method
 where TS = tabu size (for tabu search)
            or tournament size (for genetic algorithm)
  where PS = population size for genetic algorithm method only (1-60)
  where XO = crossover type for genetic algorithm method only (0-2)
  where PC = crossover probability for genetic algorithm method only
             (0-10000,eg. 100=1%)
  where PM = mutation probability for genetic algorithm method only
             (0-10000,eg. 100=1%)
 where OF = original image filename to use
 where EF = edge-only image filename to use
 where CP = compressed image filename to use
 where XF = expanded image filename to use
```

EXAMPLE INPUT (with explanation):

```
quanmain TABU 8 -8 256 256 8 8 0 200 -1 0.6 0.95 300 40 abdom1.raw
  abdom1_tsPLAP8s40.edg abdom1_tsPLAP8s40.cmp abdom1_tsPLAP8s40.exp
```

This command line says to use TABU SEARCH to adaptively compress the abdom1.raw image (8 bpp, 256 rows, 256 cols) using a kernel size of 8, 8 quantization matrices (and edgy clusters), an automatic threshold, 300 iterations, and a tabu size of 40. The last 3 arguments are for the edge, compressed, and expanded image files. Note that the "0", "200", "0.6", and "0.95" arguments for the defunct Kohonen classification scheme are no longer used.

Figure 2: Input Format and Example Input

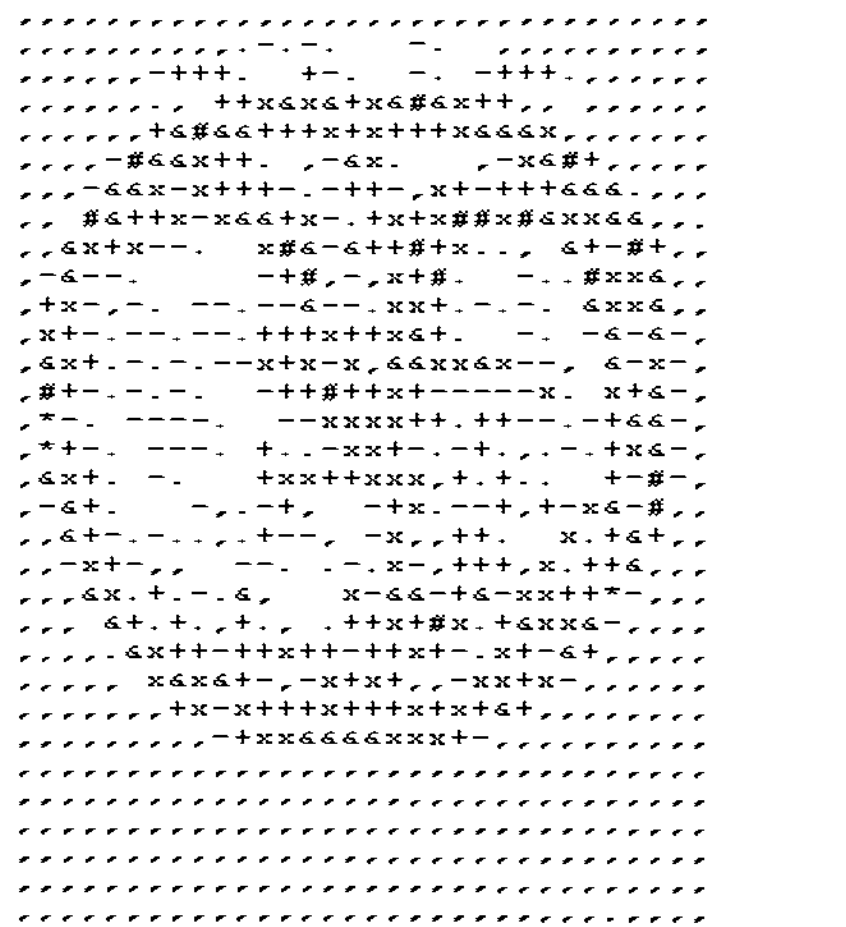
EXAMPLE OUTPUT:

```
QUANMAIN (version 1.4):
Running TABU SEARCH method!
  bitsused=8
  ksize=8
  rows=256
  cols=256
  numquants=8
  maxclust=8
  initrad=0
  kohiters=200
  threshval=-1
  learnrate=0.6000
  adjrate=0.9500
  iters=300
  tsize=40
  origfile=abdom1.raw
  edgefile=abdom1_tsPLAP8s40.edg
  compfile=abdom1_tsPLAP8s40.cmp
  expfile=abdom1_tsPLAP8s40.exp

The min, max and average of the gradients are:
  Min =        0
  Max =      255
  Avg =     45.5
  Thr =      234 (auto='.9*(Max-Avg)')
Edge detected file printed to file abdom1_tsPLAP8s40.edg!
   - - - - -
   - - - - -
```

Figure 3: Example Output

EXAMPLE OUTPUT: (continued)



```
*-*-* ITERATION 1!
**--**--** NEW BEST 0[0.10803,0.11281]
**--**--** NEW BEST 1[0.11072,0.11626]
**--**--** NEW BEST 2[0.11706,0.12414]
**--**--** NEW BEST 3[0.12495,0.12910]
**--**--** NEW BEST 4[0.13522,0.13942]
**--**--** NEW BEST 5[0.14848,0.14930]
**--**--** NEW BEST 6[0.15363,0.16080]
**--**--** NEW BEST 7[0.11007,0.11144]
    - -       .
*-*-* ITERATION 300!
**--**--** NEW BEST 3[0.11628,0.11628]
```

Figure 4: Example Output (continued)

EXAMPLE OUTPUT: (continued)

```
 6, 12, 8, 16, 16, 26, 16, 16,
 6, 20, 16, 34, 32, 34, 42, 50,
 6, 16, 40, 36, 54, 64, 76, 84,
 6, 26, 48, 66, 70, 86, 100, 118,
 30, 36, 46, 70, 92, 108, 134, 154,
 14, 40, 66, 84, 108, 136, 154, 186,
 22, 46, 74, 106, 132, 160, 190, 216,
 26, 52, 82, 118, 150, 180, 218, 256,
(0 above)
 6, 6, 10, 16, 16, 16, 16, 34,
 6, 20, 16, 16, 32, 36, 44, 64,
 14, 16, 32, 36, 66, 64, 70, 88,
 14, 32, 46, 48, 66, 88, 104, 112,
 30, 40, 80, 64, 88, 108, 132, 152,
 30, 52, 88, 96, 110, 132, 154, 176,
 30, 52, 80, 106, 128, 154, 182, 218,
 30, 76, 102, 116, 150, 178, 214, 256,
(1 above)

 4, 6, 30, 18, 20, 16, 26, 22,
 6, 14, 18, 20, 34, 42, 56, 48,
 14, 26, 42, 40, 56, 70, 74, 94,
 8, 32, 40, 56, 72, 78, 100, 110,
 16, 40, 54, 66, 88, 106, 130, 144,
 16, 48, 66, 80, 104, 130, 156, 174,
 20, 44, 78, 100, 130, 154, 184, 214,
 18, 44, 86, 128, 152, 184, 210, 256,
(6 above)
 4, 10, 10, 10, 16, 16, 22, 34,
 8, 10, 20, 28, 40, 34, 44, 64,
 10, 16, 28, 42, 48, 66, 74, 84,
 8, 34, 36, 42, 70, 82, 98, 128,
 16, 32, 52, 64, 88, 104, 130, 150,
 16, 34, 64, 82, 104, 136, 156, 182,
 20, 40, 74, 102, 130, 160, 182, 214,
 18, 54, 86, 116, 154, 182, 220, 250,
(7 above)
    original_bpp = 8.00000
    compress_bpp = 2.71071
    compress_mse = 212.37215
    compress_rms = 14.57299
    compress_snr = 24.85983
    compress_rat = 0.10904
SUCCESSFUL RUN!
```

Figure 5: Example Output (continued)

GENETIC ALGORITHM CODE SNAPSHOT:

```
randomize(0);                          /* randomize & initialize below */
if (!makerefbits(bitsused) || !makecosinematrices(DOINGFORWARD,ksize)
                           || !makecosinematrices(DOINGINVERSE,ksize))
   /* PROBLEM MAKING REF or COS MATRICES */
indivlen = ksize * ksize;              /* do GA initialization now */
done = gainit(bitsused,ksize,rows,cols,numquants,psize);
while (iter < iters && done <= 0)      /* loop thru iterations */
{  iter++;                             /* loop thru quants below */
   for (qmatrix = 0; qmatrix < numquants; qmatrix++)
   {  gatournament(numquants,qmatrix,tsize,psize,&p1,&p2,&c1,&c2);
      gacrossover(ksize,qmatrix,p1,p2,c1,c2,xover,pc);
      gamutation(ksize,qmatrix,c1,c2,pm);
      for (k = 0; k < psize; k++)      /* do tourn/xover/mutation above */
      {  if (c1 == k || c2 == k)       /* evaluate new children only */
         {  for (i = 0; i < ksize; i++)
            {  for (j = 0; j < ksize; j++)
                  quant_matrix[qmatrix][i][j] = quant_gamatrix[k][qmatrix][i][j];
            }                          /* compress & expand image below */
            if (!compressimage(bitsused,ksize,rows,cols,-qmatrix,"","",))
               /* PROBLEM WITH COMPRESSION */
            if (!expandimage(bitsused,ksize,rows,cols,-qmatrix,"",""))
               /* PROBLEM WITH EXPANSION */
            evalimages(bitsused,ksize,rows,cols,-qmatrix,original_data,
                       expanded_data,&mse,&rms,&bpp,&snr,&rat);
            a = (k + 1) * numquants; /* evaluate above, check BEST below */
            if (rat < rat_best[a + qmatrix])
               rat_best[a + qmatrix] = rat;
            if (rat < rat_best[qmatrix])
            {  for (i = 0; i < ksize; i++)
               {  for (j = 0; j < ksize; j++)
                     quant_tabubest[qmatrix][i][j] = quant_matrix[qmatrix][i][j];
               }                       /* recorded optimal child above */
               rat_best[qmatrix] = rat;
               quant_bestiter[qmatrix] = iter;
            }
         }
      }
   }
}
for (qmatrix = 0; qmatrix < numquants; qmatrix++)
{  for (i = 0; i < ksize; i++)         /* restore best matrices */
   {  for (j = 0; j < ksize; j++)
         quant_matrix[qmatrix][i][j] = quant_tabubest[qmatrix][i][j];
   }
}                                      /* do real compress/expand/eval */
if (compressimage(bitsused,ksize,rows,cols,numquants,infile,cmpfile))
{  if (expandimage(bitsused,ksize,rows,cols,numquants,cmpfile,outfile))
   {  evalimages(bitsused,ksize,rows,cols,numquants,original_data,
                 expanded_data,&mse,&rms,&bpp,&snr,&rat);
   }
}
```

Figure 6: Genetic Algorithm Code Snapshot

TABU SEARCH CODE SNAPSHOT:

```
randomize(0);                          /* randomize & initialize below */
if (!makerefbits(bitsused) || !makecosinematrices(DOINGFORWARD,ksize)
                           || !makecosinematrices(DOINGINVERSE,ksize))
   /* PROBLEM MAKING REF or COS MATRICES */
z = ref_bits[bitsused];                /* do TS initialization now */
done = tsinit(bitsused,ksize,rows,cols,numquants,tsize);
while (iter < iters && done <= 0)      /* loop thru iterations */
{  iter++;                             /* loop thru quants below */
   for (qmatrix = 0; qmatrix < numquants; qmatrix++)
   {  done = tsbest(bitsused,ksize,rows,cols,iter,qmatrix,
                    &besti,&bestj,&bestk,&bestcost);
      if (done > 0)                    /* find best move above */
         /* PROBLEM FINDING BEST MOVE */
      else if (done)                   /* do move & check BEST below */
      {  quant_tabulist[qmatrix][besti][bestj] = iter + quant_tabusize[qmatrix];
         tsmove(&bestk,z,qmatrix,besti,bestj);
         if (bestcost < rat_best[qmatrix])
         {  for (i = 0; i < ksize; i++)
            {  for (j = 0; j < ksize; j++)
                  quant_tabubest[qmatrix][i][j] = quant_matrix[qmatrix][i][j];
            }                          /* recorded optimal move above */
            rat_best[qmatrix] = bestcost;
            quant_bestiter[qmatrix] = iter;
         }
      }
   }
}
/* SAME restoration of best matrices and real compress/expand/eval as GA code */
```

BELOW IS A CODE SNAPSHOT OF THE *tsbest(...)* ROUTINE

```
z = ref_bits[bitsused];                /* some values initialized */
for (i = 0; i < ksize; i++)            /* loop thru entire kernel */
{  for (j = 0; j < ksize; j++)
   {  for (k = 0; k < 2; k++)          /* loop thru '+' and '-' moves */
      {  prevqmatval = quant_matrix[qmatrix][i][j];
         if (!tsmove(&k,-z,qmatrix,i,j))
            break;                     /* compress/expand/eval image below */
         if (!compressimage(bitsused,ksize,rows,cols,-qmatrix,"","",))
            /* PROBLEM WITH COMPRESSION */
         if (!expandimage(bitsused,ksize,rows,cols,-qmatrix,"",""))
            /* PROBLEM WITH EXPANSION */
         evalimages(bitsused,ksize,rows,cols,-qmatrix,original_data,
                    expanded_data,&mse,&rms,&bpp,&snr,&rat);
         quant_matrix[qmatrix][i][j] = prevqmatval;
         if (quant_tabulist[qmatrix][i][j] < iter || rat < rat_best[qmatrix])
         {  if (rat < *bestcost)
            {  *besti = i; *bestj = j; *bestk = k; *bestcost = rat;
            }
         }
      }
   }
}
```

Figure 7: Tabu Search Code Snapshot

TESTING AND ANALYSIS:

TEST IMAGES:

```
Abdominal 1 ..... abdom1.raw
Abdominal 2 ..... abdom2.raw
Abdominal 3 ..... abdom3.raw
Abdominal 4 ..... abdom4.raw
Abdominal 5 ..... abdom5.raw
Brain 0 ......... brain0.raw
Brain 1 ......... brain1.raw
Brain 2 ......... brain2.raw
Spine ........... spine.raw
Lena ............ lenaraw.raw
Mandrill ........ mandrill.raw
```

ANALYSIS:

* Compare recovered (expanded) images to the original image and/or lowest compressed image
* Compare the charts produced for all variations of classification, TS, and GA used.
* Make conclusions based upon all results.

KEY (for results):

```
bpp = bits per pixel
mse = mean squared error
rms = root mean squared error
snr = peak signal-to-noise ratio (in dB)
rat = cost function ratio (e.g. bpp/snr)
time = elapsed time for algorithm (in hh:mm:ss)
```

Figure 8: Testing and Analysis Preview

Masters Thesis (Winter 1998/1999)

APPENDIX C – User and Technical Manuals

Michael S. Champion (e-mail: msc0686@cs.rit.edu)

January 28, 1999

1 User Documentation

Those attempting to read this section should read the theoretical background presented in the Masters Thesis before continuing. This will provide the user with the knowledge necessary to make good choices for program arguments.

1.1 Program Description

The program *quanmain* allows 2 techniques the user may choose for edge-adaptive block quantization during the compression of small images. The key idea for each method is to find the best set of quantization tables for a given image or image type based on the relative edginess of internal blocks. To accomplish this task a genetic algorithm or tabu search algorithm may be employed. For simplicity of implementation, the software only handles 8 bit grayscale images which are 256 rows and 256 columns in dimension. The software is also only capable of using an 8 byte kernel size for the DCT transform, and the number of quantization matrices used to encode the image may only be 1, 2, 4, 8, or 16. The final outcome is the edge-detected view of the original image file, a compressed version of the image, and the expanded form of the compressed image. The following subsections explain in more detail the functionality of this program.

1.2 Input Data

The input data for both optimization algorithms includes the TABU or GENALG flag for the method selected, bits per pixel, kernel size, rows of pixels, columns of pixels, number of quantization matrices to use, the maximum number of clusters used for classification, and the threshold value used for classification. These values should all be given in this order followed by 3 additional values used for the now defunct Kohonen classification: number of Kohonen iterations, learning rate, and adjustment to learning rate. Although these 3 parameters must be filled-in, anything specified for them will not be considered during the run of the program (unless a hardcoded flag is reset). The 2 rate parameters may be substituted as cost function weights for bits per pixel and peak signal-to-noise ratio, respectively (see Figure 9 of Appendix A and Figure 2 of Appendix B).

For tabu search (TABU) the only additional parameter is the tabu size. However, for the genetic algorithm (GENALG) approach 5 additional arguments must be given in order as tournament size, population size, crossover type, crossover probability, and mutation probability. Both methods are concluded with 4 parameters which are the original image filename used for input and the edge-only filename, compressed image filename, and expanded image filename used by the program for output (see Figure 2 of Appendix B for details).

1.3 Normal Output

The normal output indicated by Figures 3, 4, and 5 of Appendix B is initially the display of program arguments along with edge-detection and block classification statistics. This is the time when the edge-detected image is written to a file. Next, the output of each iteration of the selected algorithm is displayed, showing any new best and old best pairs found for each quantization matrix. After the final iteration all quantization matrices are shown, the evaluation statistics are given, and the remaining compressed and expanded image output files are generated.

1.4 Exception Reports

The only time something should go wrong with the program is if there is an internal problem with DCT mathematics or matrix manipulation, or there is an insufficient number of command line arguments. In the first case you will get an error saying "ERROR: dct or insert went wrong!". This is coupled with a "PROBLEM DURING RUN!" message if the problem occurred during the run of the algorithm, or it is coupled with "PROBLEM DURING CLASSIFICATION!" if the problem occurred during classification. In both cases you will get a "INTERNAL ERROR: There may be a USAGE problem, so try the following" message afterward followed by the usage requirements given in Figure 2 of Appendix B. Segmentation faults and arithmetic exception errors could result if incorrect command line arguments are given. If either of these errors should occur the user should re-examine the command line versus the actual usage (the *core* file generated should be removed).

1.5 Program Limitations

As mentioned previously this program is not capable of processing anything other than 8 bit images that are 256 rows by 256 columns in dimension. Also, for quantization the program only accepts an 8 byte kernel size and 1, 2, 4. 8, or 16 for the number of quantization matrices. All other directions given in Figure 2 of Appendix B should be followed explicitly.

1.6 Command Sequence

There is an example of user arguments given at the bottom of Figure 2 in Appendix B. It should be clear after reading this example how the program should behave. To run the program using a genetic algorithm instead of tabu search simply replace "TABU" with "GENALG" and replace the tabu size with a valid tournament size. Also, the 4 extra parameters for population size, crossover type, crossover probability, and mutation probability must be given in order right after the tournament size (but before the 4 filenames are given).

Any problems or questions encountered during usage or modification of this program should be forwarded to the author:

Name: Michael Champion

Address: msc0686@cs.rit.edu

Telephone: 716-724-4000

2 Technical Documentation

Those attempting to read this section should read the functional description presented in the Masters Thesis before continuing. This will provide the programmer with the knowledge necessary to make good choices for program modifications. In addition, it would be very helpful to read and understand the code and internal documentation referenced in Appendix N before reading further. The module and function descriptions allow a good understanding of what the program does and how it does it.

2.1 Program Description

This program adaptively quantizes an image based upon the relative edginess of each 64 byte square block in the image. To do this it employs the edge-detection algorithm given in the lower part of Figure 7 in Appendix A. Afterward it requires the use of either greedy algorithm represented in Figure 8 of Appendix A. The implementation of both genetic algorithm and tabu search approaches is shown in Figures 6 and 7 of Appendix B. This program is only designed to run in a UNIX environment, and the command line used to compile and link each module together is given at the beginning of each source file referenced in Appendix N.

2.2 Historical Development of the Program

Included in the heading description of each module referenced in Appendix N is a timeline giving a brief description of the changes which took place during the development cycle on the program. The following is a more detailed account of changes which took place.

Initially the edge-detection, Kohonen SOM, and other classification algorithms were built in parallel with the image bit-packing, compression, and expansion routines. These functions were tested separately such that an edge form of the image could be generated, and such that the image could be successfully compressed and expanded within a relatively low degree of error. This phase also included the minor testing and debugging of slave routines for DCT calculation and matrix manipulation. Note that all of the associated routines were created with the intention of increasing the maximum number of quantization matrices in the near future.

Once this stage was complete, the genetic algorithm (GA) and tabu search (TS) code was built around the image compression, expansion, and evaluation functions already created. All routines had to be slightly revised for the correct usage of multiple quantization matrices. Once the GA and TS code proved to be working properly some testing was done to find out if each method did try to find the optimum quantization matrix for each image block classification. Afterward, much debugging code was stripped from each module and the evaluation routine was made more robust.

If more information is needed please contact the author at the address or phone number listed at the end of the previous (User Documentation) section.

2.3 Overall System and Program Structure

Figure 1 of Appendix B gives a combined look at the system organization and general flow of data within the program. A more detailed understanding of the diagram may be obtained from the functional description portion of the Masters Thesis document. There are 8 different modules with various functions in each one. The *quanmain* file parses the command line arguments and sets the proper values to be passed on to the *quanclas* module. The *classifyimage* routine in this file calls the *edgedetect* function of the *quanedge* module to return an edge-detected form of the image which will have its kernel-sized regions labeled as belonging to a class based on its relative edginess.

Once classes are assigned to each image block the original image data is passed on to the *quanadap* module where either *gacompress* or *tscompress* is used to find the optimum set of quantization matrices, minimizing the cost function given in Figure 9 of Appendix A. To compress, expand, and evaluate an image the *quanpack* and *quanmisc* modules must be accessed. It should be noted that the routines in these 3 files occasionally need the functions contained within the *quanmake* and *quanmatr* modules in order to complete their tasks. However, with the notable exception of the *quanmisc* module which contains miscellaneous functions that did not fall under any particular category, each module contains functions which accomplish similar tasks. There are other helper routines not mentioned here which assist in each task. The internal documentation of the code referenced in Appendix N contains more details on each of the functions.

2.4 Description of Key Data Structures

All block packing storage is represented by dynamically created and freed 1 dimensional arrays within the *quanclas* module. All other arrays are created at compile time (see the *quanmain.h* include file referenced in Appendix N for more details). All temporary image storage is represented by 1 dimensional arrays. Floating-point arrays are used during calculation within the *dct2d* function, as well as some matrix manipulation functions. However, normally short-integer arrays are used throughout the program for data representation.

Two dimensional arrays are never used in the program, but 3 and 4 dimensional arrays are used quite often. The tabu list for every move in each quantization matrix is represented by a 3 dimensional array, as are the storage matrix and the best tabu matrix found for each of the quantization matrices (also used for the genetic algorithm method). The current quantization matrices being manipulated are also 3 dimensional arrays, but in order to accommodate each population member for the genetic algorithm method a 4 dimensional array was used. These are all represented as short-integer arrays, and they are created upon compilation of the program (i.e. they are non-dynamic in size).

Extra 1 dimensional floating-point arrays were necessary for the best cost ratio for each quantization matrix, the cosine matrices (normal and transpose), and classification weights. Additional 1 dimensional short-

integer arrays were needed for reference bits, the Gaussian smoothing filter, the Laplacian operators, the Sobel operators, the best tabu (or genetic algorithm) iteration for each quantization matrix, and the tabu size for each quantization matrix. The image matrices were represented by unsigned-character arrays for ease in access to files. These global variables were also created at compile time.

Visit the code reference in Appendix N for module descriptions and more detailed information on how some of the global variables mentioned are used and modified.

2.5 Built-in Maintenance Aids

There is no explicit way to flip a switch in the code and have elaborate debugging information shown. However, any programmer wishing to use leftover debugging code can search the code for "debug" variables. No timing variables or routines are used within the program, so timing should be done outside the program. If internal timing is desired, suitable locations may be found within the *quanadap* module. Also, "printf" statements can be used anywhere within the program to uncover cryptic problems with code changes. Note that the "rogertst" flag of the *quanmain.h* file can be set to examine DCT calculations before a certain number of iterations is reached (see Appendix N for source code reference and DCT dissection).

2.6 Testing/Acceptance Criteria

The simplest way to test the program is to find an 8 bit image that is 256 rows by 256 columns in size and plug it into the example command line given in Figure 2 of Appendix B. This command line can be modified and reconstructed based upon the requirements desired. Results can be compared with other valid configurations using several different image files and file types. When a program run is complete an image manipulation tool like *osiris* (or some other raw image viewer) may be used to view the expanded image(s) for subjective analysis.

Masters Thesis (Winter 1998/1999)

APPENDIX D – Original Image Data

Michael S. Champion (e-mail: msc0686@cs.rit.edu)

January 28, 1999

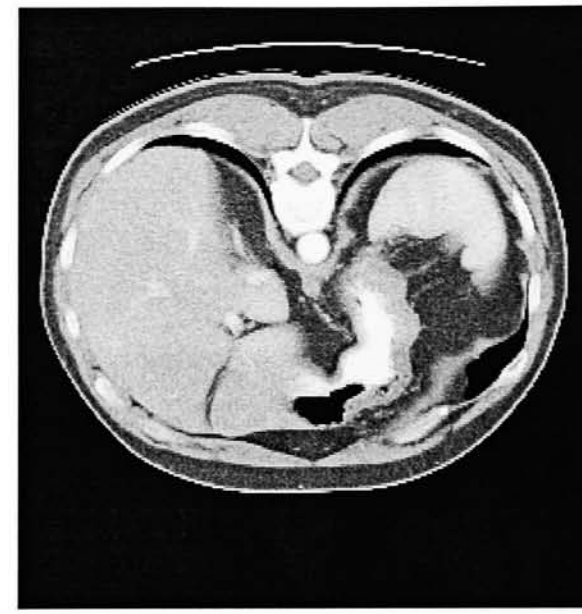


Figure 1: Abdominal 1 – Original Raw Data

Figure 2: Abdominal 2 – Original Raw Data

Figure 3: Abdominal 3 – Original Raw Data

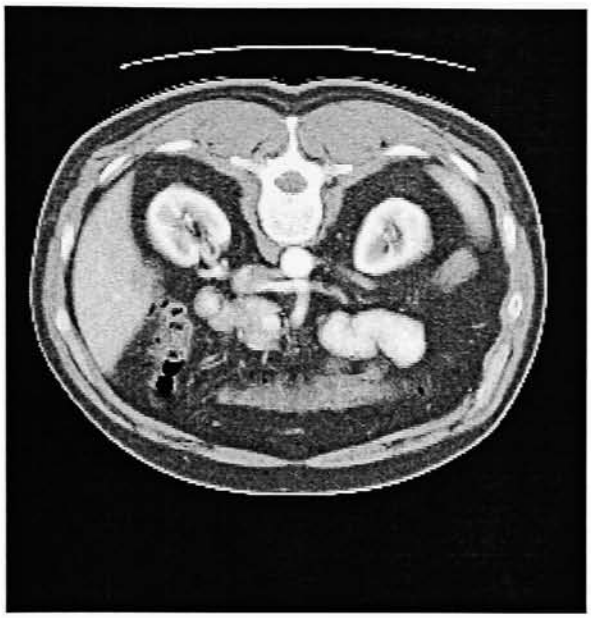


Figure 4: Abdominal 4 - Original Raw Data

Figure 5: Abdominal 5 - Original Raw Data

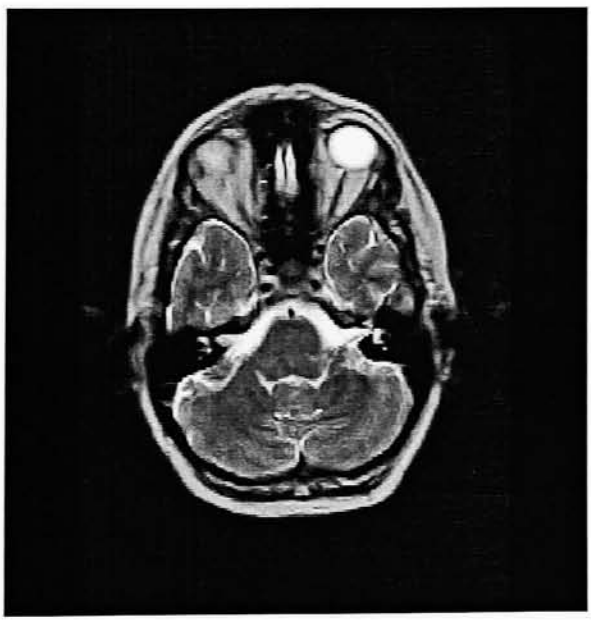


Figure 6: Brain 0 (top view) - Original Raw Data

Figure 7: Brain 1 (side view) - Original Raw Data

Figure 8: Brain 2 (side view) – Original Raw Data

Figure 9: Spine – Original Raw Data

Figure 10: Lena – Original Raw Data

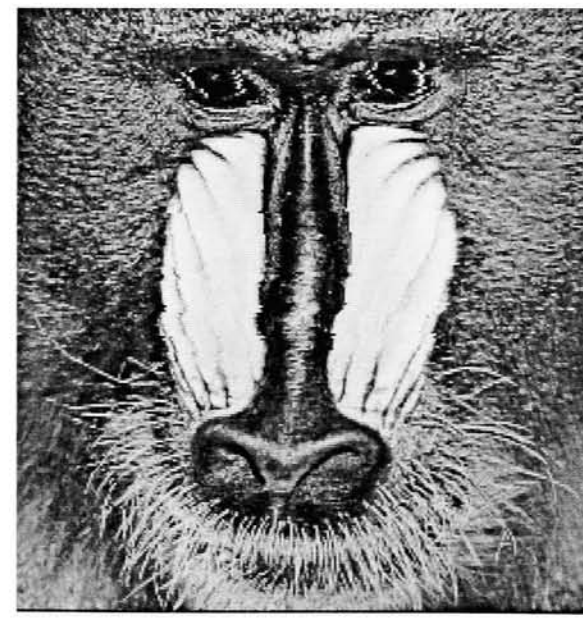


Figure 11: Mandrill – Original Raw Data

Masters Thesis (Winter 1998/1999)

APPENDIX E - Recovered (Expanded) Image Data

Michael S. Champion (e-mail: msc0686@cs.rit.edu)

January 28, 1999

Figure 1: Abdominal 1 - Expanded Raw Data

Figure 2: Abdominal 2 - Expanded Raw Data

Figure 3: Abdominal 3 - Expanded Raw Data

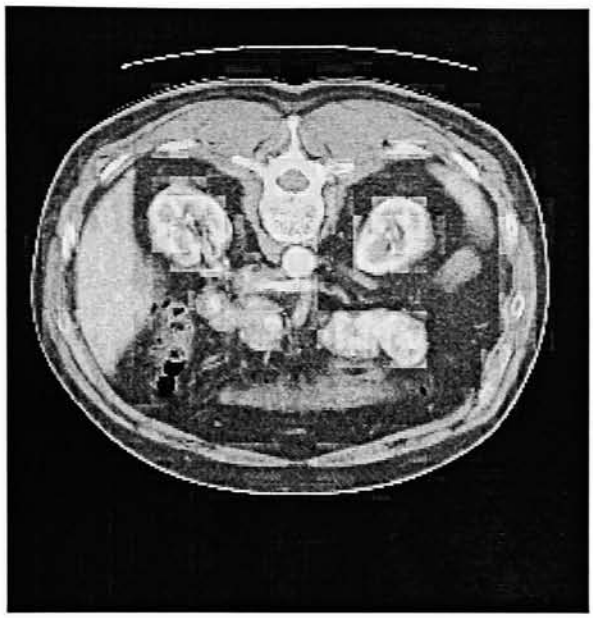


Figure 4: Abdominal 4 - Expanded Raw Data

Figure 5: Abdominal 5 - Expanded Raw Data

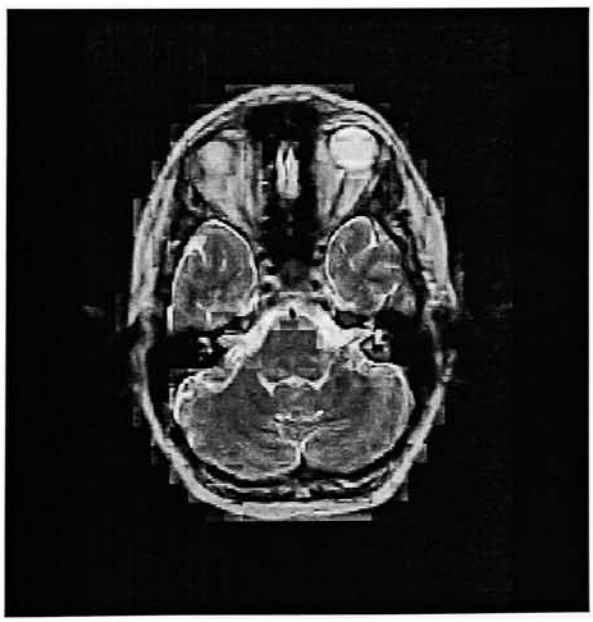


Figure 6: Brain 0 (top view) - Expanded Raw Data

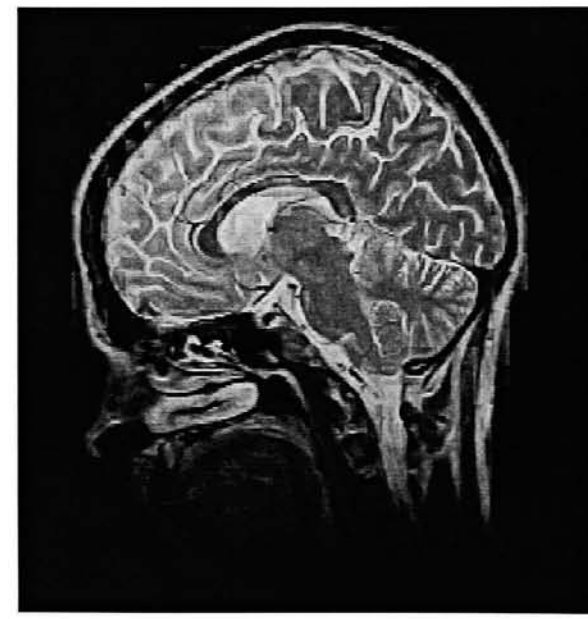


Figure 7: Brain 1 (side view) - Expanded Raw Data

Figure 8: Brain 2 (side view) – Expanded Raw Data

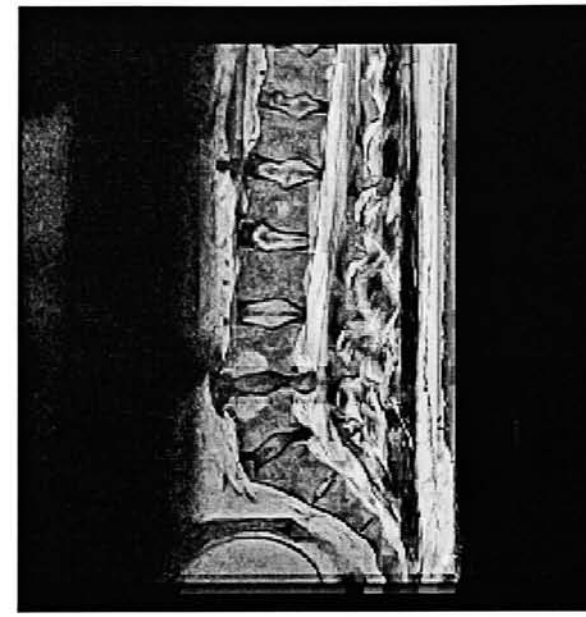


Figure 9: Spine – Expanded Raw Data

Figure 10: Lena – Expanded Raw Data

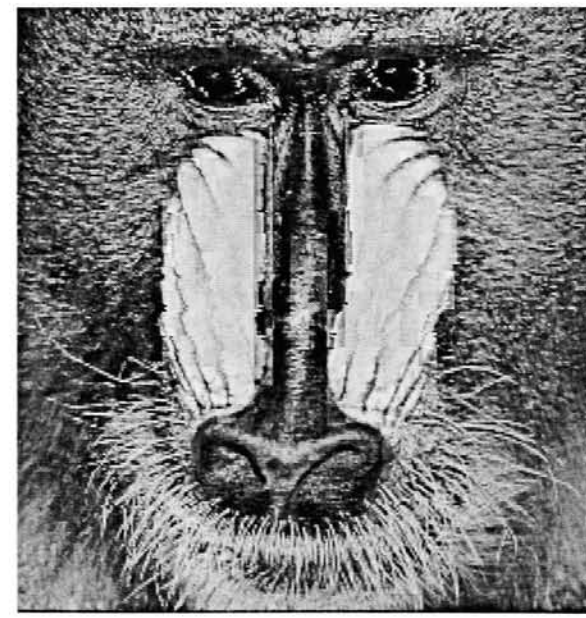


Figure 11: Mandrill – Expanded Raw Data

Masters Thesis (Winter 1998/1999)

APPENDIX F – Edge-Detected (Sobel operator) Image Data

Michael S. Champion (e-mail: msc0686@cs.rit.edu)

January 28, 1999

Figure 1: Abdominal 1 – Sobel Edge Data

Figure 2: Abdominal 2 – Sobel Edge Data

Figure 3: Abdominal 3 – Sobel Edge Data

Figure 4: Abdominal 4 - Sobel Edge Data

Figure 5: Abdominal 5 - Sobel Edge Data

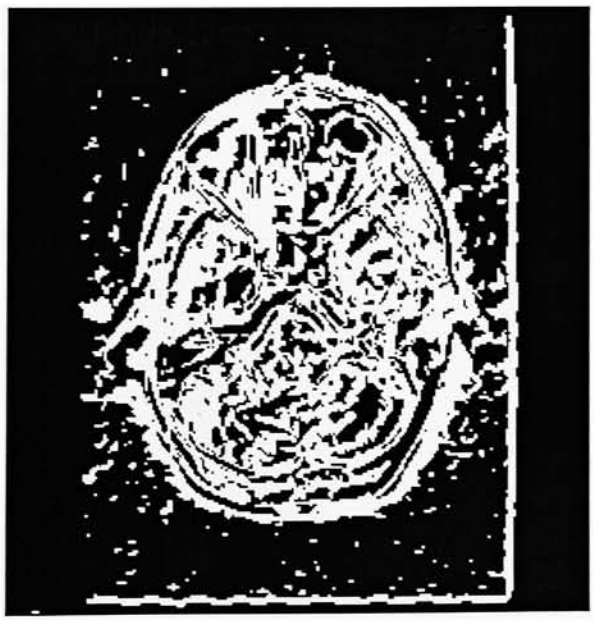


Figure 6: Brain 0 (top view) - Sobel Edge Data

Figure 7: Brain 1 (side view) - Sobel Edge Data

Figure 8: Brain 2 (side view) – Sobel Edge Data

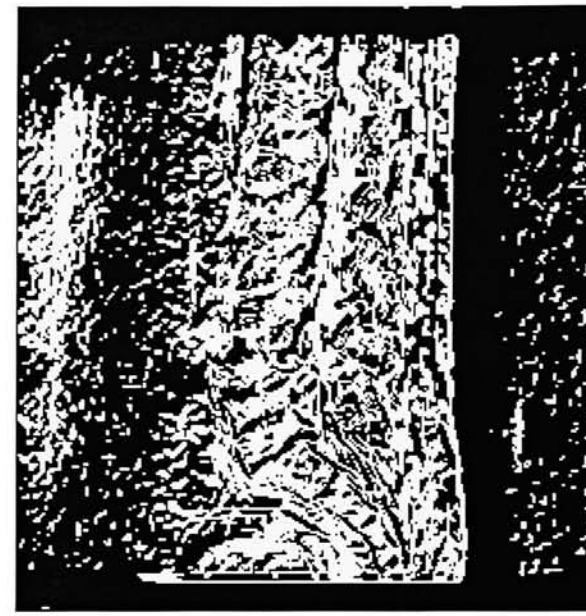


Figure 9: Spine – Sobel Edge Data

Figure 10: Lena – Sobel Edge Data

Figure 11: Mandrill – Sobel Edge Data

Masters Thesis (Winter 1998/1999)

APPENDIX G – Edge-Detected (Laplacian operator) Image Data

Michael S. Champion (e-mail: msc0686@cs.rit.edu)

January 28, 1999

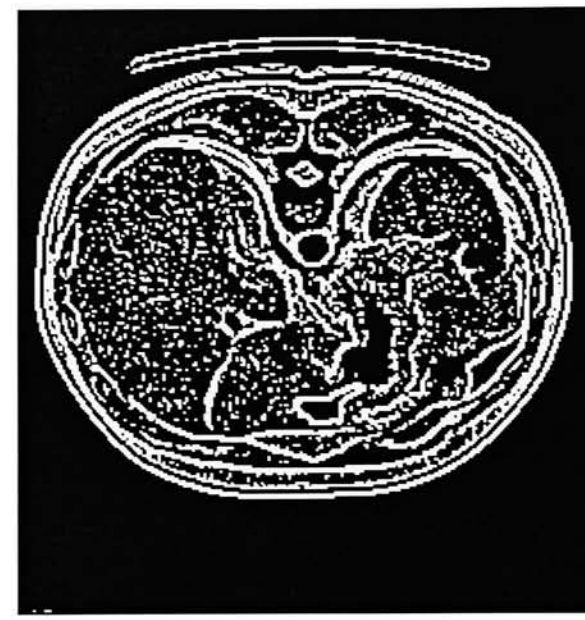


Figure 1: Abdominal 1 - Laplace Edge Data

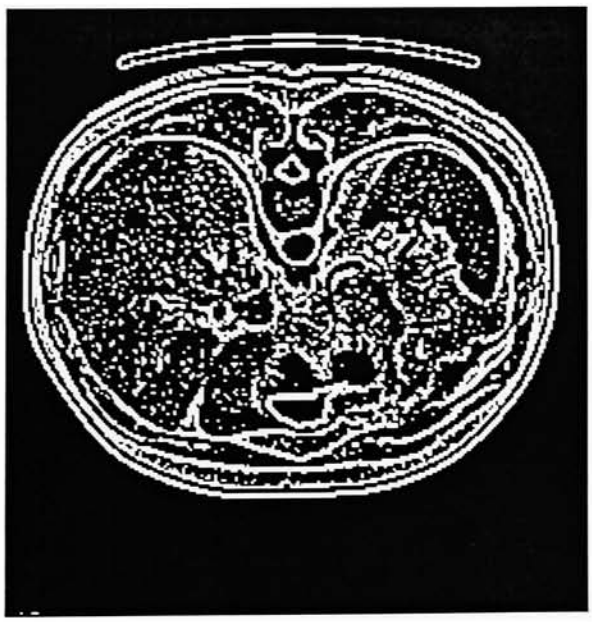


Figure 2: Abdominal 2 - Laplace Edge Data

Figure 3: Abdominal 3 - Laplace Edge Data

Figure 4: Abdominal 4 – Laplace Edge Data

Figure 5: Abdominal 5 – Laplace Edge Data

Figure 6: Brain 0 (top view) – Laplace Edge Data

Figure 7: Brain 1 (side view) – Laplace Edge Data

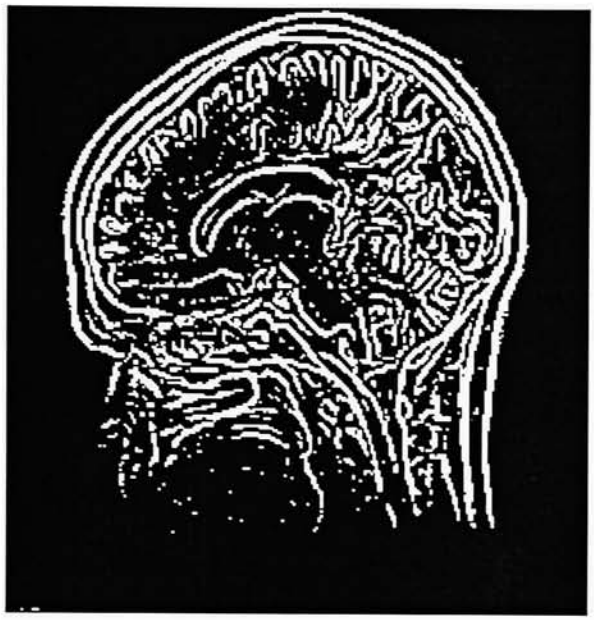


Figure 8: Brain 2 (side view) – Laplace Edge Data

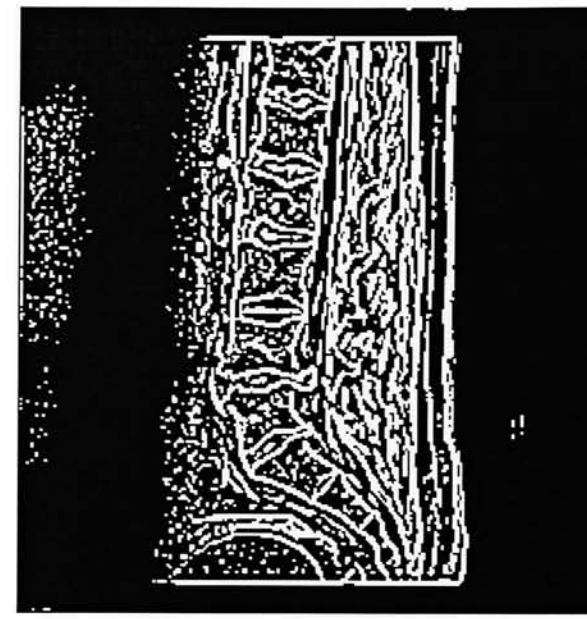


Figure 9: Spine – Laplace Edge Data

Figure 10: Lena – Laplace Edge Data

Figure 11: Mandrill – Laplace Edge Data

Masters Thesis (Winter 1998/1999)

APPENDIX H – Genetic Algorithm Results

(varying crossover type, tournament size, and population size)

using Abdominal 1 image

Michael S. Champion (e-mail: msc0686@cs.rit.edu)

January 28, 1999

Figure 1: GA Results – control image

Figure 2: GA Results – using uniform crossover

Figure 3: GA Results – using one-point crossover

Figure 4: GA Results - using tournament size 5 (population size 60)

Figure 5: GA Results - using tournament size 15 (population size 60)

Figure 6: GA Results - using tournament size 25 (population size 60)

Figure 7: GA Results - using tournament size 35 (population size 60)

Figure 8: GA Results - using tournament size 55 (population size 60)

Figure 9: GA Results - using population size 20 (tournament size 15)

Figure 10: GA Results - using population size 30 (tournament size 15)

Figure 11: GA Results - using population size 40 (tournament size 15)

Figure 12: GA Results – using population size 50 (tournament size 15)

Figure 13: GA Results – using population size 60 (tournament size 15)

Masters Thesis (Winter 1998/1999)

APPENDIX I – Genetic Algorithm Results

(varying crossover rate, mutation rate, edge operation,

and number of quantization matrices)

using Abdominal 1 image

Michael S. Champion (e-mail: msc0686@cs.rit.edu)

January 28, 1999

Figure 1: GA Results – control image

Figure 2: GA Results – using 70% crossover rate

Figure 3: GA Results – using 80% crossover rate

Figure 4: GA Results - using 90% crossover rate

Figure 5: GA Results - using 2% mutation rate

Figure 6: GA Results - using 6% mutation rate

Figure 7: GA Results - using 8% mutation rate

Figure 8: GA Results - using edge transitions

Figure 9: GA Results - using Sobel operator

Figure 10: GA Results - using edge transitions and Sobel operator

Figure 11: GA Results - using 1 quantization matrix

Figure 12: GA Results – using 2 quantization matrices

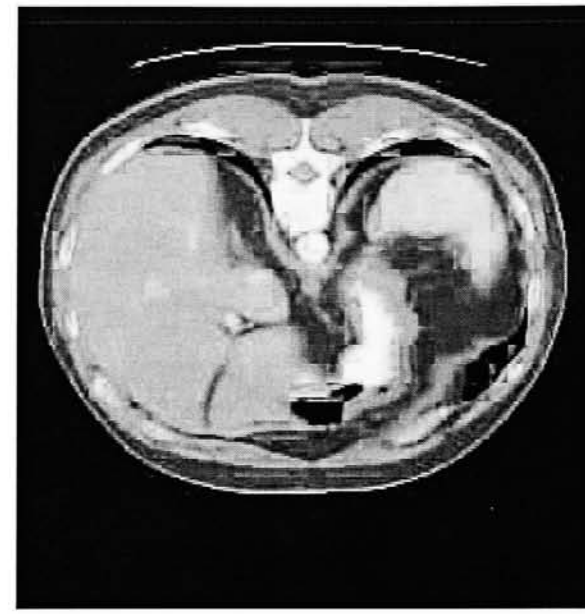


Figure 13: GA Results – using 4 quantization matrices

Masters Thesis (Winter 1998/1999)

APPENDIX J – Genetic Algorithm Results

(used on same type image plus other images)

using Abdominal 1 image

Michael S. Champion (e-mail: msc0686@cs.rit.edu)

January 28, 1999

Figure 1: GA Results – Abdominal 1 image (control image)

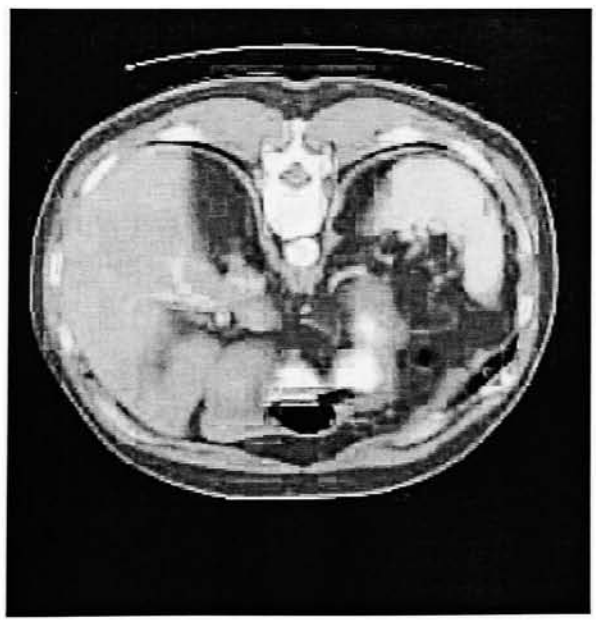


Figure 2: GA Results – Abdominal 2 image using quant. matrices of Abdominal 1

Figure 3: GA Results – Abdominal 3 image using quant. matrices of Abdominal 1

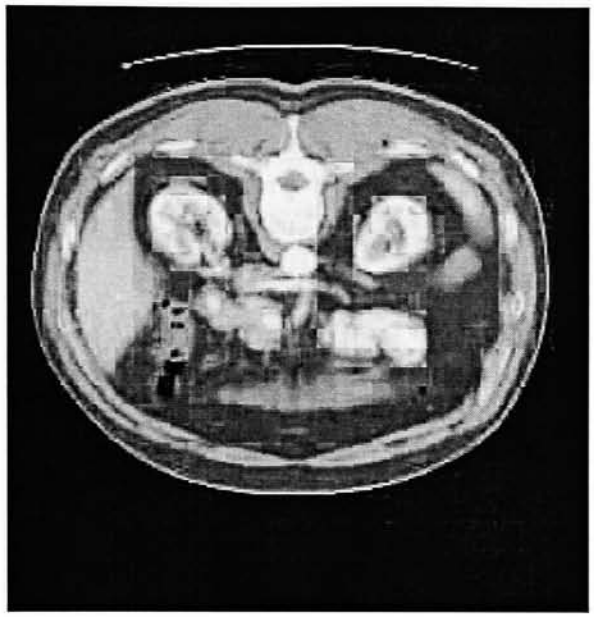


Figure 4: GA Results - Abdominal 4 image using quant. matrices of Abdominal 1

Figure 5: GA Results - Abdominal 5 image using quant. matrices of Abdominal 1

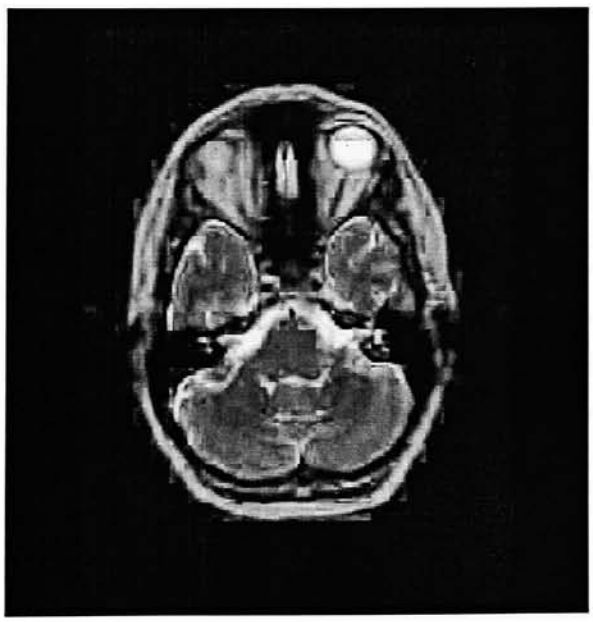


Figure 6: GA Results - Brain 0 (top view) image

Figure 7. GA Results - Brain 1 (side view) image

Figure 8: GA Results - Brain 2 (side view) image

Figure 9: GA Results - Spine image

Figure 10: GA Results - Lena image

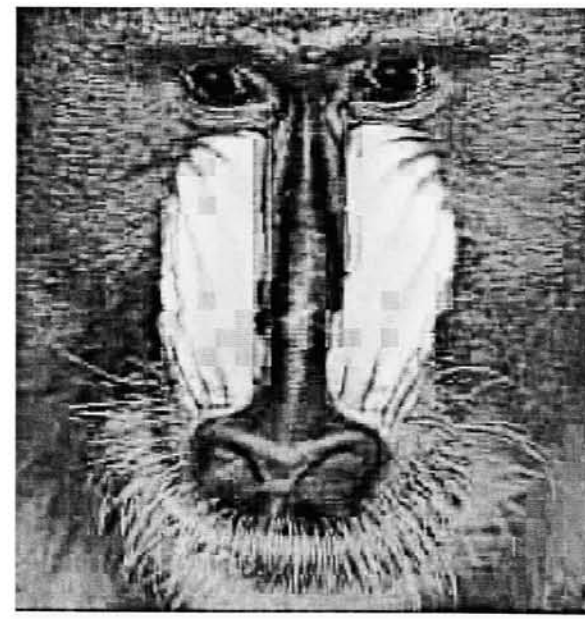


Figure 11: GA Results - Mandrill image

Masters Thesis (Winter 1998/1999)

APPENDIX K – Tabu Search Results

(varying tabu size, edge operator,

and number of quantization matrices)

using Abdominal 1 image

Michael S. Champion (e-mail: msc0686@cs.rit.edu)

January 28, 1999

Figure 1: TS Results – control image

Figure 2: TS Results – using tabu size 10

Figure 3: TS Results – using tabu size 25

Figure 4: TS Results - using tabu size 55

Figure 5: TS Results - using tabu size 70

Figure 6: TS Results - using tabu size 85

Figure 7: TS Results - using tabu size 100

Figure 8: TS Results – using edge transitions

Figure 9: TS Results – using Sobel operator

Figure 10: TS Results – using edge transitions and Sobel operator

Figure 11: TS Results – using 1 quantization matrix

Figure 12: TS Results – using 2 quantization matrices

Figure 13: TS Results – using 4 quantization matrices

Masters Thesis (Winter 1998/1999)

APPENDIX L – Tabu Search Results

(used on same type image plus other images)

using Abdominal 1 image

Michael S. Champion (e-mail: msc0686@cs.rit.edu)

January 28, 1999

Figure 1: TS Results – Abdominal 1 image (control image)

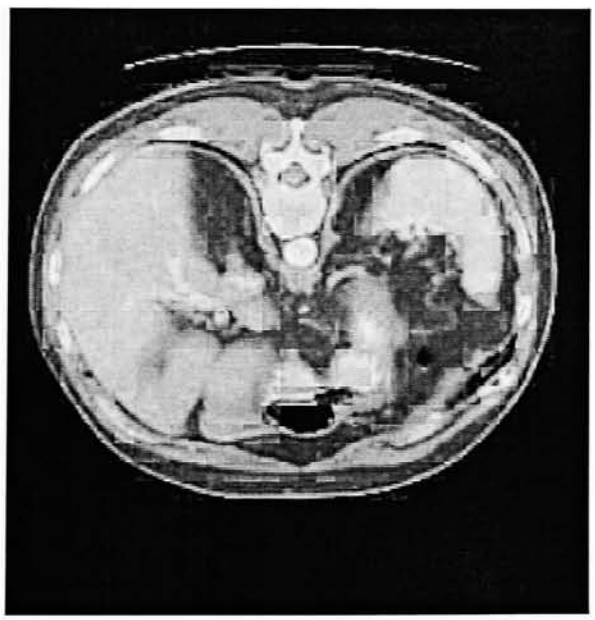


Figure 2: TS Results – Abdominal 2 image using quant. matrices of Abdominal 1

Figure 3: TS Results – Abdominal 3 image using quant. matrices of Abdominal 1

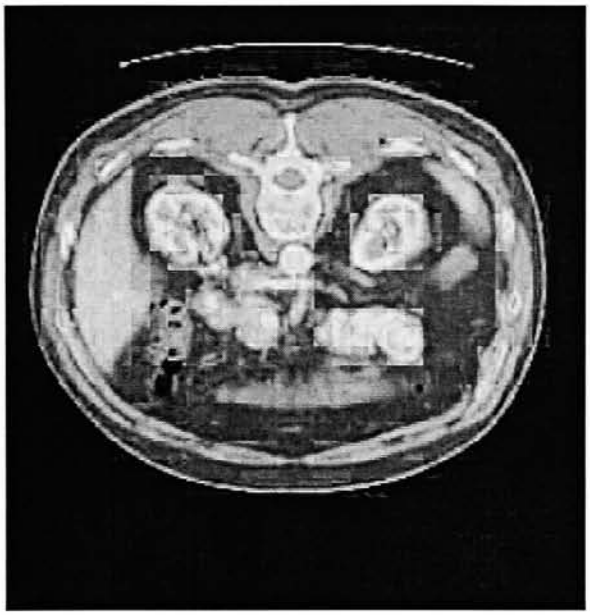


Figure 4: TS Results - Abdominal 4 image using quant. matrices of Abdominal 1

Figure 5: TS Results - Abdominal 5 image using quant. matrices of Abdominal 1

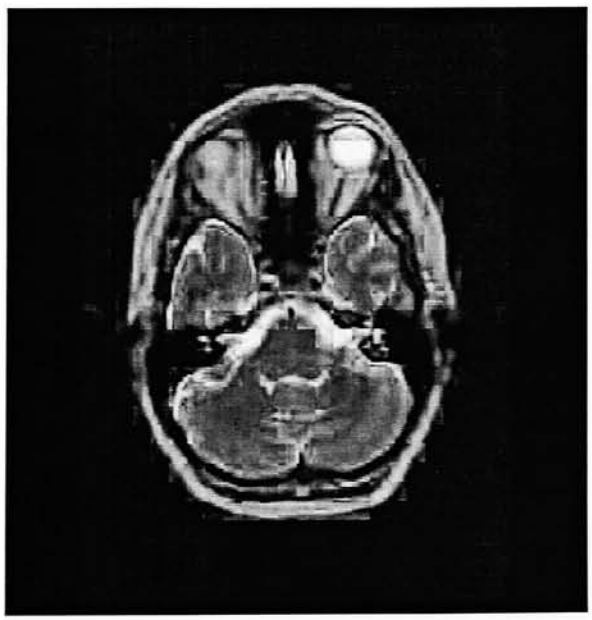


Figure 6: TS Results - Brain 0 (top view) image

Figure 7: TS Results - Brain 1 (side view) image

Figure 8: TS Results - Brain 2 (side view) image

Figure 9: TS Results - Spine image

Figure 10: TS Results - Lena image

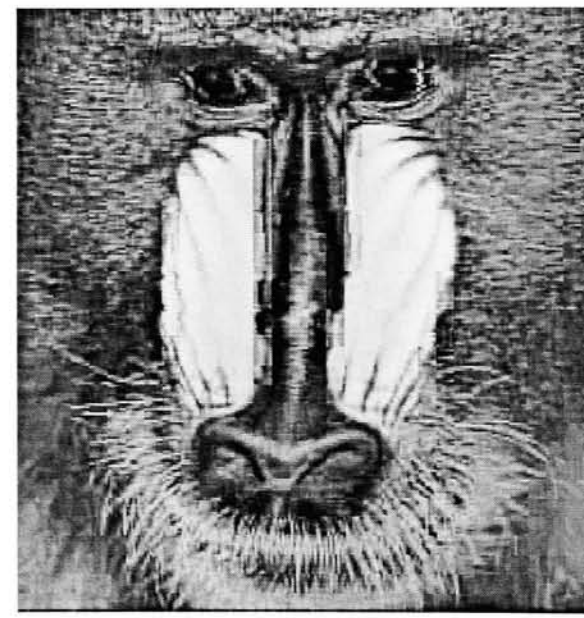


Figure 11: TS Results - Mandrill image

Masters Thesis (Winter 1998/1999)

APPENDIX M – Results, Analysis, and Conclusions

Michael S. Champion (e-mail: msc0686@cs.rit.edu)

January 28, 1999

GENETIC ALGORITHM RESULTS:

The following results were obtained with the following parameters unless otherwise indicated:

* % blackness (not % b/w transition)
* Laplacian edge operator (not Sobel)
* 8 quantization matrices
* 45 for tournament size
* 60 for population size
* Two-point crossover
* 100% for crossover rate
* 4% for mutation rate
* 300 total iterations
* Using ABDOMINAL 1 image

NORMAL (CONTROL)
bpp = 2.65419
mse = 219.97702
rms = 14.83162
snr = 24.70703
rat = 0.10743
time = 00:33:12

WITH % B/W TRANSITION
bpp = 2.63542
mse = 211.86824
rms = 14.55569
snr = 24.87015
rat = 0.10597
time = 00:31:58

WITH SOBEL OPERATOR
bpp = 2.64304
mse = 215.54861
rms = 14.68157
snr = 24.79535
rat = 0.10659
time = 00:32:48

WITH % TRANS. & SOBEL OP.
bpp = 2.64522
mse = 212.59074
rms = 14.58049
snr = 24.85536
rat = 0.10642
time = 00:32:07

WITH 1 QUANT. MATRIX
bpp = 2.60938
mse = 230.66287
rms = 15.18759
snr = 24.50103
rat = 0.10650
time = 00:17:51

WITH 2 QUANT. MATRICES
bpp = 2.66614
mse = 208.60765
rms = 14.44326
snr = 24.93750
rat = 0.10691
time = 00:27:57

WITH 4 QUANT. MATRICES
bpp = 2.67917
mse = 216.80716
rms = 14.72437
snr = 24.77007
rat = 0.10816
time = 00:30:07

Figure 1: GA Solution – evaluation results

GENETIC ALGORITHM RESULTS: *(continued)*

WITH 70% CROSSOVER RATE
bpp = 2.64168
mse = 216.81990
rms = 14.72481
snr = 24.76981
rat = 0.10665
time = 00:32:41

WITH 80% CROSSOVER RATE
bpp = 2.64287
mse = 212.51376
rms = 14.57785
snr = 24.85693
rat = 0.10632
time = 00:33:01

WITH 90% CROSSOVER RATE
bpp = 2.63103
mse = 214.92644
rms = 14.66037
snr = 24.80791
rat = 0.10606
time = 00:31:58

WITH 2% MUTATION RATE
bpp = 2.62675
mse = 220.66753
rms = 14.85488
snr = 24.69342
rat = 0.10637
time = 00:32:42

WITH 6% MUTATION RATE
bpp = 2.62538
mse = 213.24216
rms = 14.60281
snr = 24.84207
rat = 0.10568
time = 00:32:55

WITH 8% MUTATION RATE
bpp = 2.60356
mse = 221.01433
rms = 14.86655
snr = 24.68660
rat = 0.10546
time = 00:32:30

WITH UNIFORM CROSSOVER
bpp = 2.66568
mse = 210.66011
rms = 14.51413
snr = 24.89498
rat = 0.10708
time = 00:32:26

WITH ONE-POINT CROSSOVER
bpp = 2.64163
mse = 216.74446
rms = 14.72224
snr = 24.77132
rat = 0.10664
time = 00:32:22

Figure 2: GA Solution – evaluation results (continued)

GENETIC ALGORITHM RESULTS: (continued)

WITH TOURN. SIZE 5
bpp = 2.69029
mse = 217.92439
rms = 14.76226
snr = 24.74775
rat = 0.10871
time = 00:38:50

WITH TOURN. SIZE 15	WITH TOURN. SIZE 35
bpp = 2.66592	bpp = 2.64201
mse = 221.00200	mse = 219.51711
rms = 14.86614	rms = 14.81611
snr = 24.68684	snr = 24.71612
rat = 0.10799	rat = 0.10689
time = 00:36:46	time = 00:33:59
WITH TOURN. SIZE 25	WITH TOURN. SIZE 55
bpp = 2.65268	bpp = 2.65050
mse = 219.88107	mse = 214.36505
rms = 14.82839	rms = 14.64121
snr = 24.70893	snr = 24.81926
rat = 0.10736	rat = 0.10679
time = 00:35:16	time = 00:32:00

(results with tournament size 15)

WITH POP. SIZE 20	WITH POP. SIZE 40
bpp = 2.65034	bpp = 2.64233
mse = 213.90675	mse = 215.86284
rms = 14.62555	rms = 14.69227
snr = 24.82856	snr = 24.78902
rat = 0.10675	rat = 0.10659
time = 00:29:57	time = 00:34:01
WITH POP. SIZE 30	WITH POP. SIZE 50
bpp = 2.62613	bpp = 2.66388
mse = 214.67618	mse = 215.55028
rms = 14.65183	rms = 14.68163
snr = 24.81297	snr = 24.79532
rat = 0.10584	rat = 0.10743
time = 00:34:19	time = 00:35:13

Figure 3: GA Solution – evaluation results (continued)

GENETIC ALGORITHM RESULTS: *(continued)*

(results based on quant. matrices found for ABDOMINAL 1)

ABDOMINAL 2	ABDOMINAL 4
bpp = 2.65625	bpp = 2.65625
mse = 178.63503	mse = 154.41550
rms = 13.36544	rms = 12.42640
snr = 25.61114	snr = 26.24389
rat = 0.10371	rat = 0.10121
time = 00:00:05	time = 00:00:05
ABDOMINAL 3	**ABDOMINAL 5**
bpp = 2.65625	bpp = 2.65625
mse = 158.84094	mse = 149.45261
rms = 12.60321	rms = 12.22508
snr = 26.12118	snr = 26.38577
rat = 0.10169	rat = 0.10067
time = 00:00:07	time = 00:00:07

(results based on other images)

BRAIN 0 (top view)	SPINE
bpp = 2.63655	bpp = 2.57411
mse = 96.09785	mse = 368.59204
rms = 9.80295	rms = 19.19875
snr = 28.30367	snr = 22.46534
rat = 0.09315	rat = 0.11458
time = 00:36:08	time = 00:32:24
BRAIN 1 (side view)	**LENA**
bpp = 2.63535	bpp = 2.64784
mse = 72.00020	mse = 101.82414
rms = 8.48529	rms = 10.09079
snr = 29.55747	snr = 28.05230
rat = 0.08916	rat = 0.09439
time = 00:32:57	time = 00:33:11
BRAIN 2 (side view)	**MANDRILL**
bpp = 2.61229	bpp = 2.61702
mse = 105.85138	mse = 280.23267
rms = 10.28841	rms = 16.74015
snr = 27.88384	snr = 23.65562
rat = 0.09368	rat = 0.11063
time = 00:33:17	time = 00:32:16

Figure 4: GA Solution - evaluation results (continued)

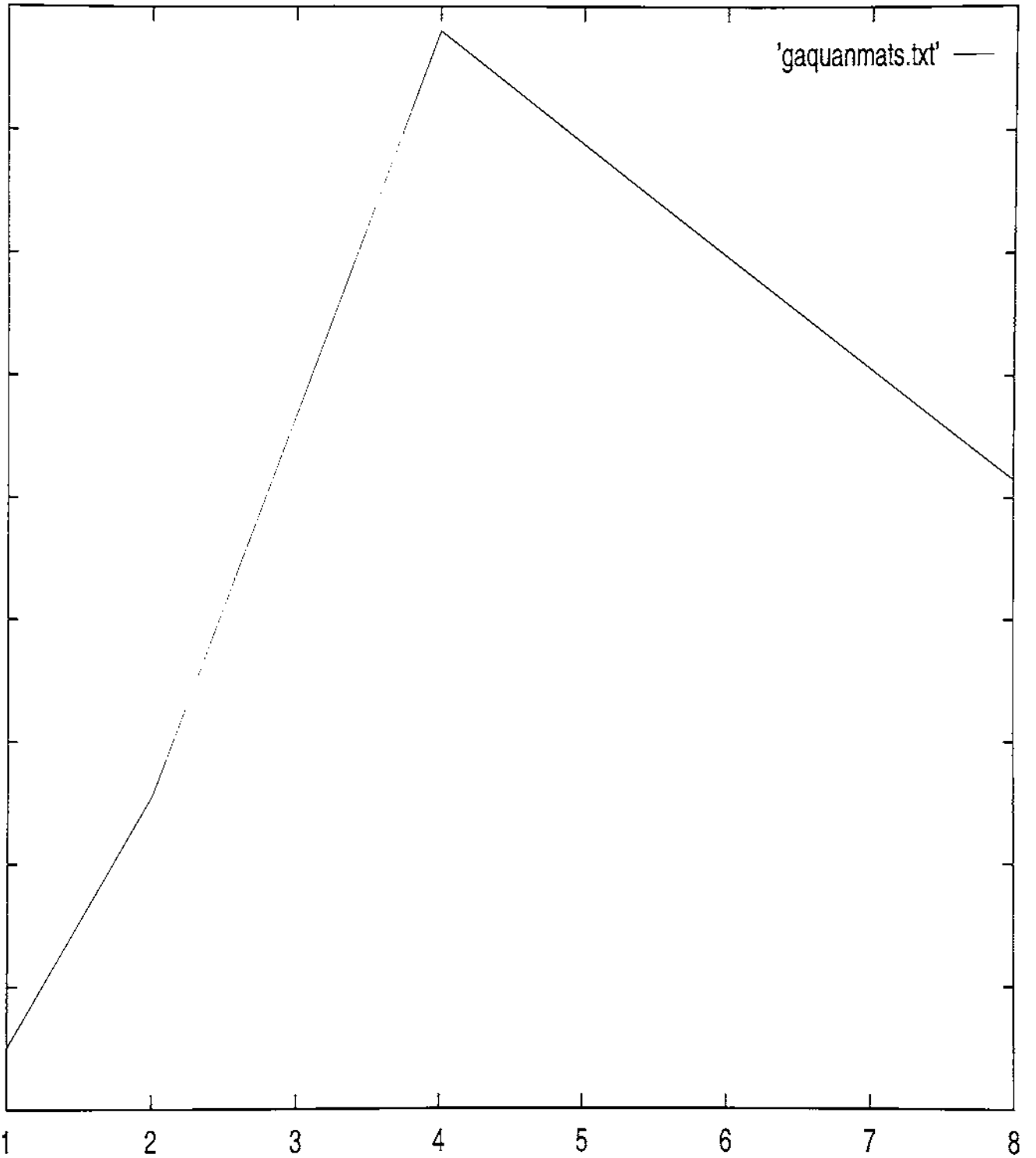


Figure 5: GA Solution – cost ratio vs. number of quant. matrices

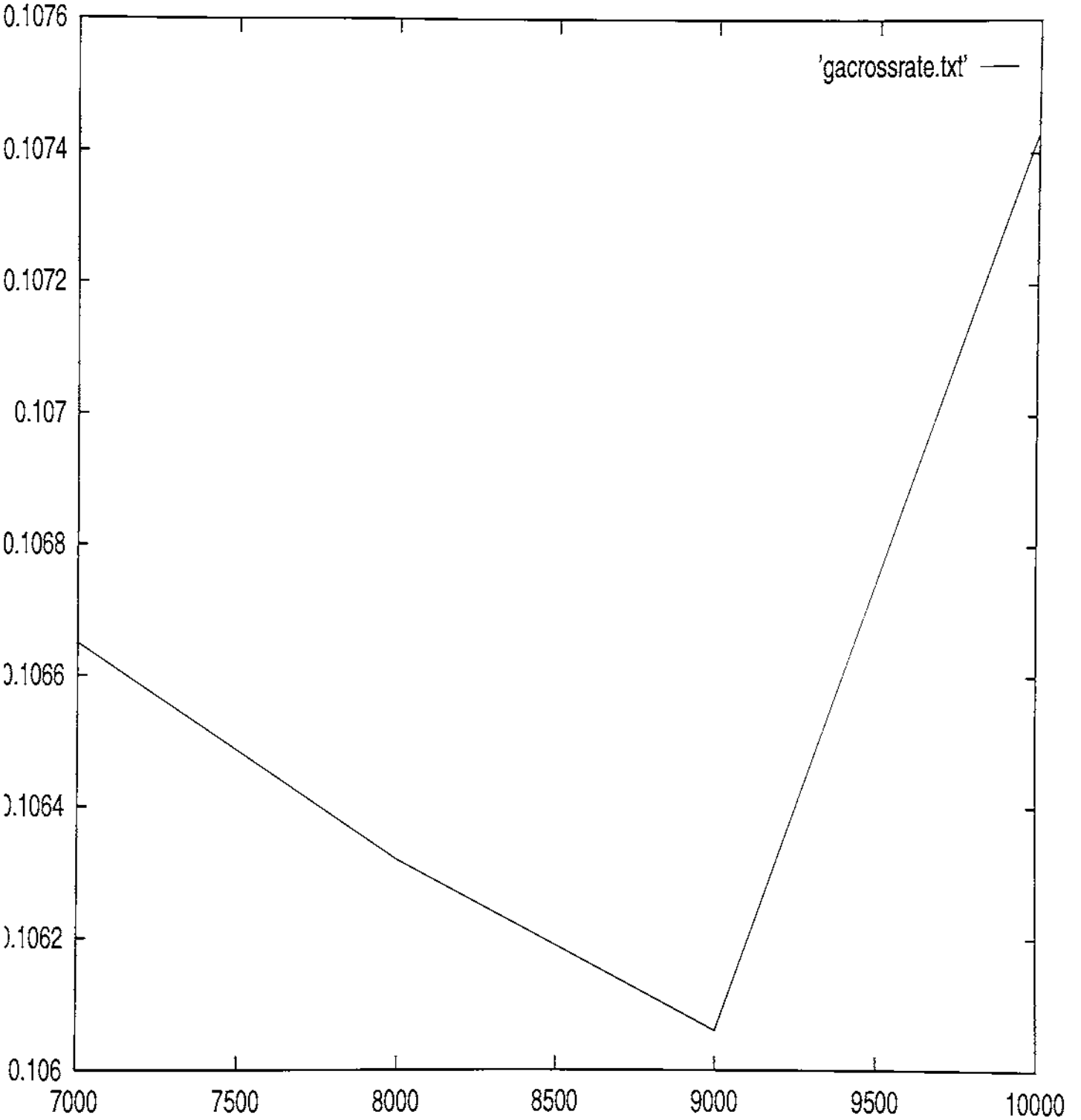


Figure 6: GA Solution – cost ratio vs. crossover rate (x10000)

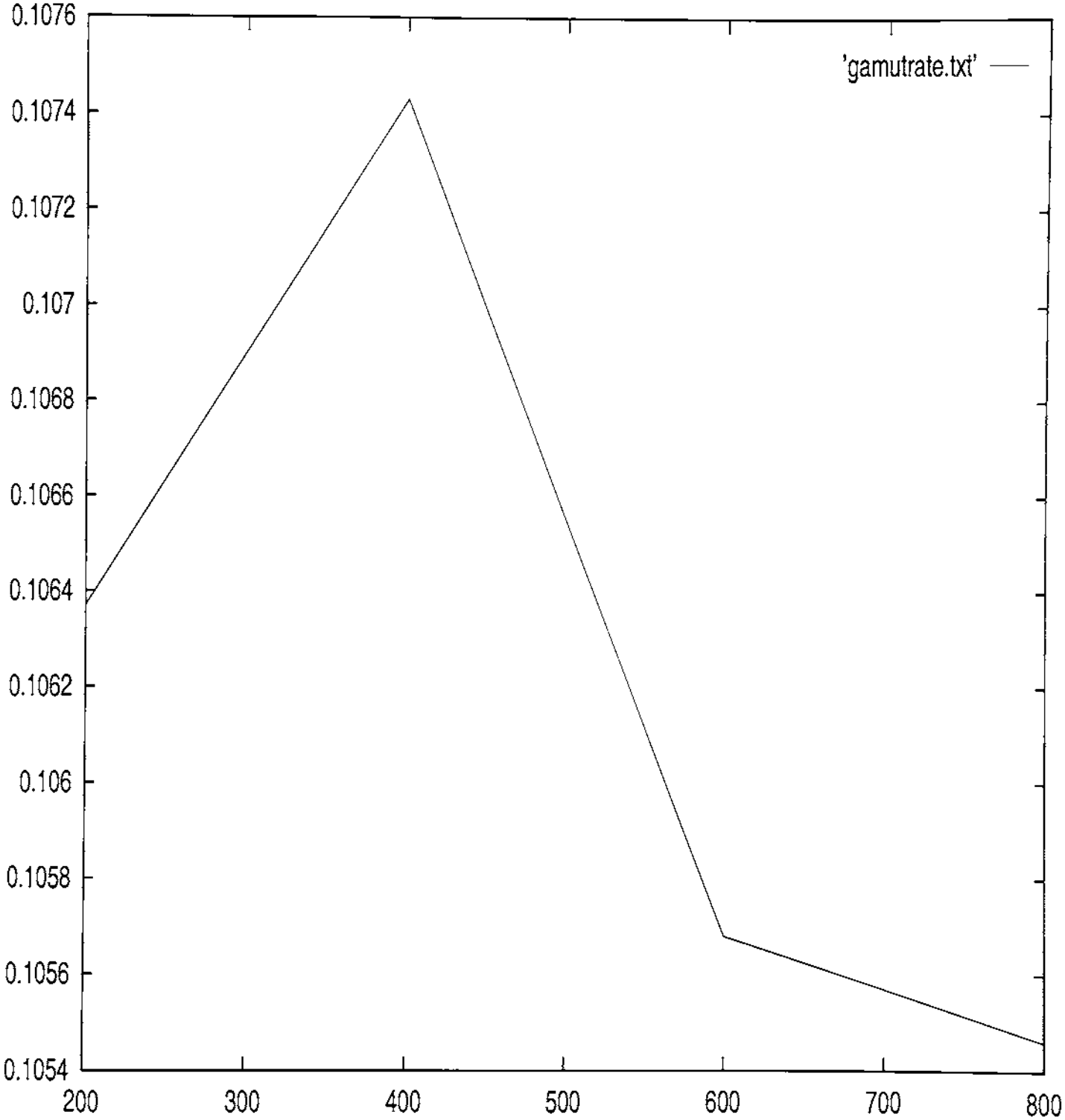


Figure 7: GA Solution – cost ratio vs. mutation rate (x10000)

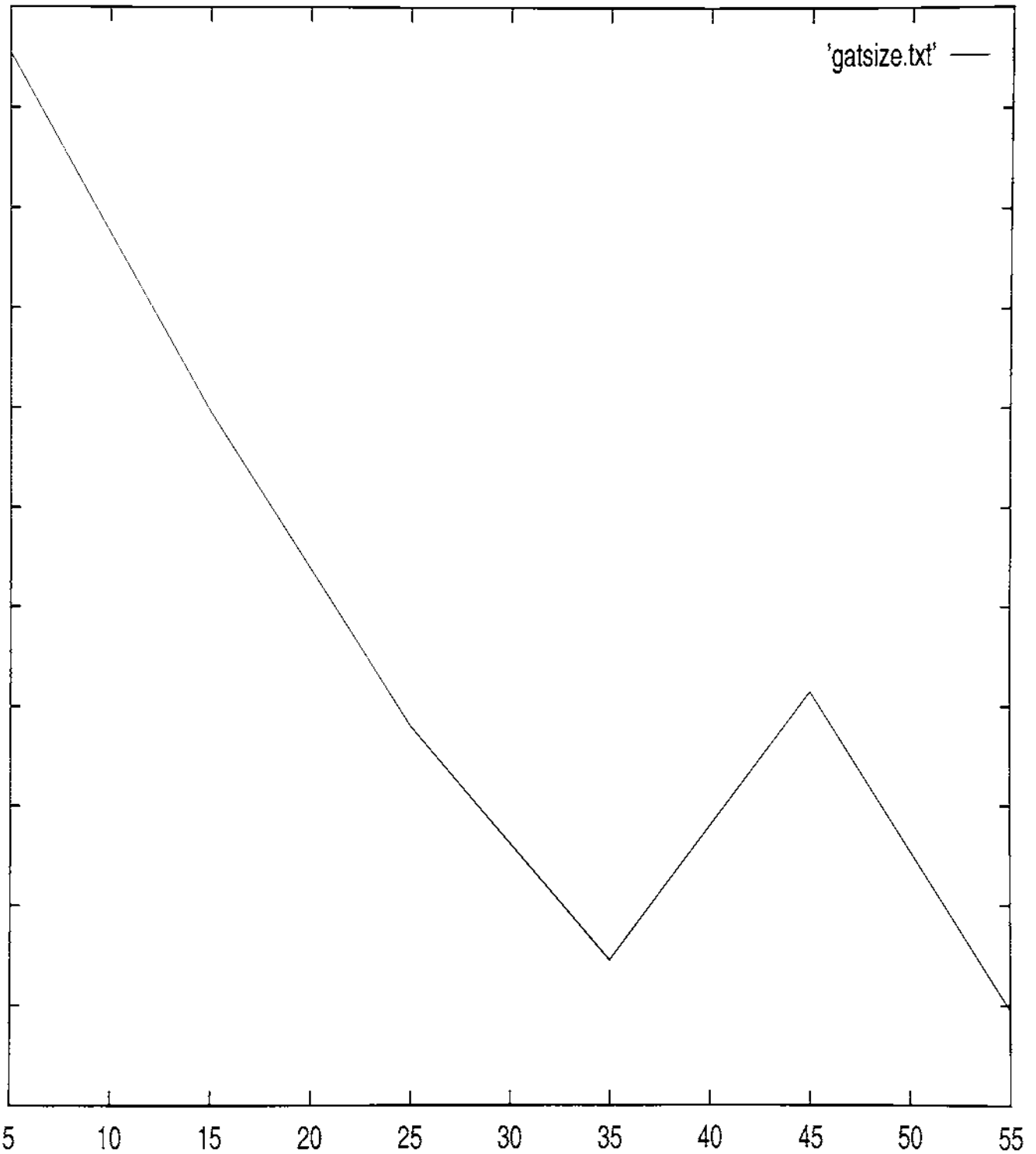


Figure 8: GA Solution – cost ratio vs. tournament size

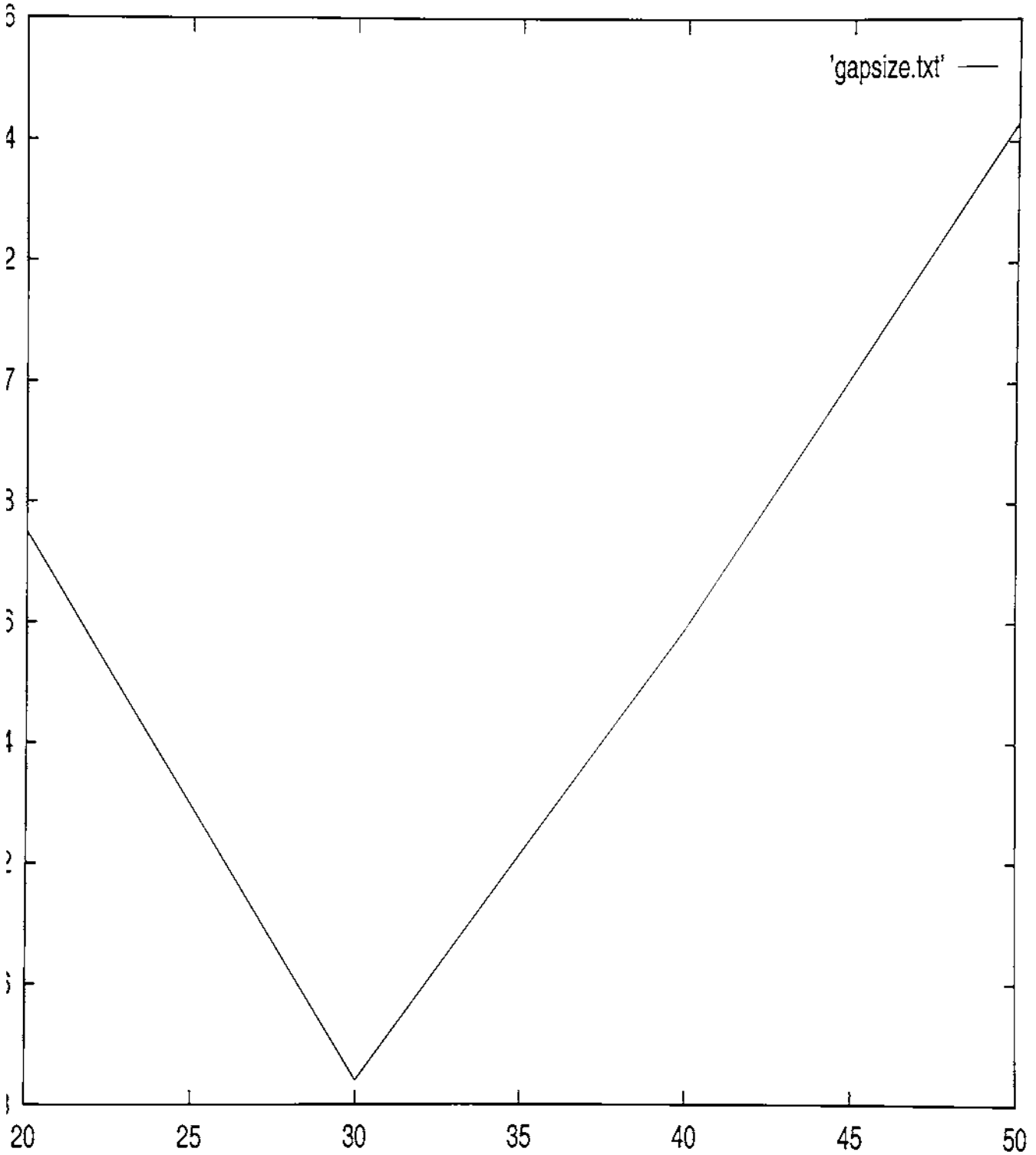


Figure 9: GA Solution – cost ratio vs. population size

TABU SEARCH RESULTS:

The following results were obtained with the following parameters unless otherwise indicated:

* % blackness (not % b/w transition)
* Laplacian edge operator (not Sobel)
* 8 quantization matrices
* 40 for tabu size
* 300 total iterations
* Using ABDOMINAL 1 image

NORMAL (CONTROL)
bpp = 2.71071
mse = 212.37215
rms = 14.57299
snr = 24.85983
rat = 0.10904
time = 32:57:00

WITH % B/W TRANSITION
bpp = 2.72234
mse = 201.39476
rms = 14.19136
snr = 25.09032
rat = 0.10850
time = 29:56:00

WITH SOBEL OPERATOR
bpp = 2.69260
mse = 208.83032
rms = 14.45096
snr = 24.93287
rat = 0.10799
time = 52:23:00

WITH % TRANS. & SOBEL OP.
bpp = 2.67787
mse = 207.14636
rms = 14.39258
snr = 24.96803
rat = 0.10725
time = 49:44:00

WITH 1 QUANT. MATRIX
bpp = 2.68750
mse = 219.31738
rms = 14.80937
snr = 24.72007
rat = 0.10872
time = 47:23:00

WITH 2 QUANT. MATRICES
bpp = 2.69897
mse = 208.58205
rms = 14.44237
snr = 24.93803
rat = 0.10823
time = 27:45:00

WITH 4 QUANT. MATRICES
bpp = 2.71942
mse = 204.15784
rms = 14.28838
snr = 25.03114
rat = 0.10864
time = 28:20:00

Figure 10: TS Solution – evaluation results

TABU SEARCH RESULTS:

WITH TABU SIZE 10	WITH TABU SIZE 70
bpp = 2.73743	bpp = 2.69690
mse = 216.47998	mse = 206.28282
rms = 14.71326	rms = 14.36255
snr = 24.77663	snr = 24.98617
rat = 0.11048	rat = 0.10794
time = 32:58:00	time = 29:53:00
WITH TABU SIZE 25	**WITH TABU SIZE 85**
bpp = 2.74377	bpp = 2.70970
mse = 208.05185	mse = 206.91350
rms = 14.42400	rms = 14.38449
snr = 24.94909	snr = 24.97292
rat = 0.10997	rat = 0.10851
time = 33:07:00	time = 52:06:00
WITH TABU SIZE 55	**WITH TABU SIZE 100**
bpp = 2.72305	bpp = 2.69566
mse = 208.67505	mse = 209.30913
rms = 14.44559	rms = 14.46752
snr = 24.93610	snr = 24.92292
rat = 0.10920	rat = 0.10816
time = 49:48:00	time = 51:48:00

(results based on quant. matrices found for ABDOMINAL 1)

ABDOMINAL 2	ABDOMINAL 4
bpp = 2.39062	bpp = 2.39062
mse = 275.70868	mse = 234.71967
rms = 16.60448	rms = 15.32056
snr = 23.72630	snr = 24.42531
rat = 0.10076	rat = 0.09787
time = 00:00:03	time = 00:00:04
ABDOMINAL 3	**ABDOMINAL 5**
bpp = 2.39062	bpp = 2.39062
mse = 233.65111	mse = 228.43417
rms = 15.28565	rms = 15.11404
snr = 24.44513	snr = 24.54319
rat = 0.09780	rat = 0.09740
time = 00:00:03	time = 00:00:03

Figure 11: TS Solution – evaluation results (continued)

TABU SEARCH RESULTS: ***(continued)***

(results based on other images)

BRAIN 0 (top view)
bpp = 2.75595
mse = 91.50017
rms = 9.56557
snr = 28.51658
rat = 0.09664
time = 31:02:00

BRAIN 1 (side view)
bpp = 2.72485
mse = 69.56815
rms = 8.34075
snr = 29.70670
rat = 0.09173
time = 31:03:00

BRAIN 2 (side view)
bpp = 2.67810
mse = 99.23703
rms = 9.96178
snr = 28.16407
rat = 0.09509
time = 33:08:00

SPINE
bpp = 2.72832
mse = 389.48999
rms = 19.73550
snr = 22.22584
rat = 0.12275
time = 33:33:00

LENA
bpp = 2.71362
mse = 96.74309
rms = 9.83581
snr = 28.27460
rat = 0.09597
time = 29:51:00

MANDRILL
bpp = 2.70041
mse = 241.80478
rms = 15.55007
snr = 24.29615
rat = 0.11115
time = 49:05:00

Figure 12: TS Solution -- evaluation results (continued)

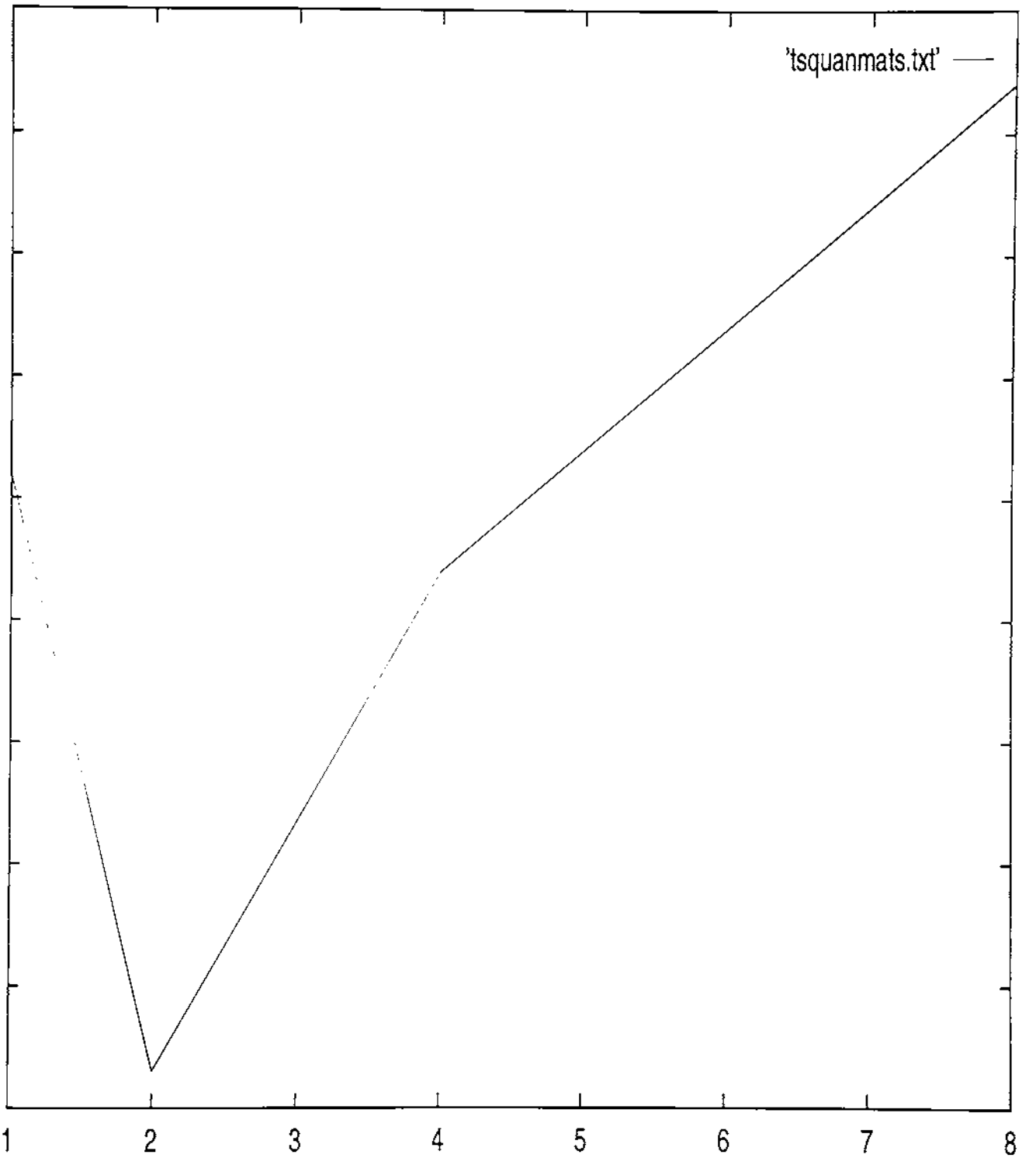


Figure 13: TS Solution – cost ratio vs. number of quant. matrices

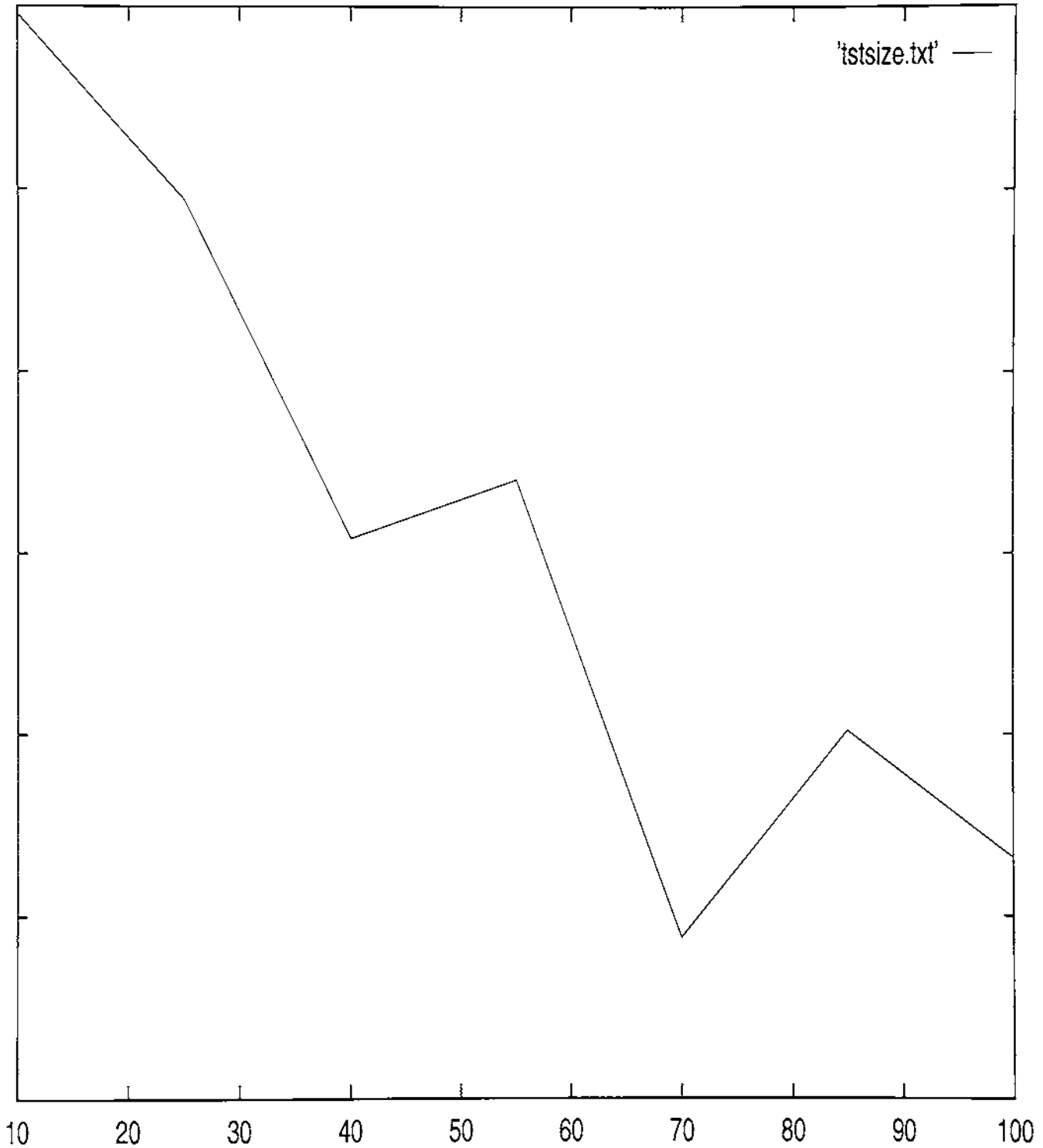


Figure 14: TS Solution – cost ratio vs. tabu size

ANALYSIS:

GENETIC ALGORITHM RESULTS: (using ABDOMINAL 1 image)

* performed best using percentage of edge-transitions and a Laplacian operator for block classification
* performed best with one-point crossover, a population size of 30, a tournament size of 55, a crossover rate of 0.90, and a mutation rate of 0.08 (approx.)
* performed best using 1 quantization matrix for image
* subjectively the one-point crossover, tournament size of 55, and mutation rate of 0.08 usage appear to show light areas darker than they really are (2-point & 35 tourn. size arguably best to use), but the variations on other parameters appears to result in about the same quality
* subjectively using 1 quantization matrix looks the worst

TABU SEARCH RESULTS: (using ABDOMINAL 1 image)

* performed best using percentage of edge-transitions and Sobel operator for block classification
* performed best with a tabu size of 70 (approx.)
* performed best using 2 quantization matrices for image
* subjectively the usage of a tabu size of 10 appears to show light areas darker than they actually are, but the variations on other parameters appears to results in about the same quality
* subjectively using 1 quantization matrix looks the best (although each usage looked very close in quality)

COMBINED RESULTS:

* GA outperforms TS in finding the best cost ratio (in a shorter amount of time) for all of the test images
* the application of 8 quantization matrices, found by the GA and TS algorithms for ABDOMINAL 1, to the other 4 ABDOMINAL images produced a slightly better cost ratio for each image (TS outperformed GA in this case only)
* subjectively it is easy to see blockiness in edge areas of all images in Appendix D-G combined, but the GA does appear to do little better (if not equally as good) than TS in the edge areas of all test images (TS does do better around the lightest part of the Mandrill's nose)

Figure 15: Analysis

ANALYSIS: *(continued)*

COMPUTATIONAL COMPLEXITY:

* Genetic Algorithm (GA):

```
initialization time = (P*Q*(3*S))                                    +
main loop time = (I*(Q*(T+(2*(K*K))+(2*3*S))))                       +
completing time = ((Q*(K*K))+(3*S))                                  =
----(adding these together to get total time below)----
total time = (P*Q*(3*S))+
             (I*(Q*(T+(2*(K*K))+(2*3*S))))+
             (Q+(3*S))
-----(asymptotic time below governed by main loop)-----
asymptotic time = O(I*(Q*(T+(2*(K*K))+(2*3*S))))
                = O(I*Q*S)
```

* Tabu Search (TS):

```
initialization time = (Q*(3*S))                                      +
main loop time = (I*(Q*(K*K)*(2*3*S)))                               +
completing time = ((Q*(K*K))+(3*S))                                  =
----(adding these together to get total time below)----
total time = (Q*(3*S))+
             (I*(Q*(K*K)*(2*3*S)))+
             (Q+(3*S))
-----(asymptotic time below governed by main loop)-----
asymptotic time = O(I*(Q*(K*K)*(2*3*S)))
                = O(I*Q*K*K*S)
```

KEY:
- K = kernel size
- I = # iterations
- Q = # quantization matrices
- P = population size
- T = tournament size
- S = image size (rows * columns)

Figure 16: Analysis (continued)

CONCLUSIONS:

* Both GA and TS methods are obviously blocky & blurry with respect to the originals, but each appears less blocky around edges with respect to lowest-compressed images.
* Both methods work well when applying set of quantization matrices, created for a given image, to similar images.
* GA better than TS in cost and time, and GA appears only marginally better subjectively.
* Using percentage of edge-transitions with either operator works best, although Laplacian edges look better, for classification (for either GA or TS).
* For GA, both objectively and subjectively, the two-point crossover, mutation rate 0.06, crossover rate 0.90, tournament size 30, and population size 30 worked best.
* For TS, both objectively and subjectively, a tabu size of about 70 worked best.
* It is too difficult to tell what number of quantization tables works best both objectively and subjectively, so the alternate cost function (with weights) was tried.

------(results with alternate cost function below)------

* Results indicate that raising the PSNR weight makes the compressibility (BPP) worse while holding the quality (PSNR) at around same value. Raising the BPP weight makes the PSNR value worse. This implies that quality and compressibility may be weighted based on the alternate cost function given in Appendix A, Figure 9.
* It appears to be the case that the more edgy the image, the more quantization tables are required to get better objective and subjective results.

Figure 17: Conclusions

Masters Thesis (Winter 1998/1999)

APPENDIX N – Source Code and DCT Dissection

(visit *www.cs.rit.edu/~msc0686/codesamp.html* for source code)

Michael S. Champion (e-mail: msc0686@cs.rit.edu)

January 28, 1999

```
QUANMAIN (version 1.4)
Running TABU SEARCH method!
  bitsused=8
  ksize=8
  rows=256
  cols=256
  numquants=8
  maxclust=8
  initrad=0
  kohiters=1
  threshval=-1
  learnrate=0.0000
  adjrate=0.0000
  iters=5
  tsize=2
  origfile=mandrill.raw
  edgefile=mand_ts.edg
  compfile=mand_ts.cmp
  expfile=mand_ts.exp

The min, max and average of the gradients are:
  Min =        0
  Max =      255
  Avg =     83.8
  Thr =      237 (auto='.9*(Max-Avg)')
Edge detected file printed to file mand_ts.edg!
-+++x+xxx+x+#&x&#&xxx+-+xxx++-++
.++xx&x#&&x&xxx&+-#xxx&&x&*xxx++
-&xxxx&x++-+++&&&*+x&xx+&#xx&+&+
+xxx&x&+&*-x#x-+x+##x+&+-+x&+xxx
+-&+xx-+-&#&*&&xx&&#&&x+&xx++x#-
++xx+x+xx&+xx+#&xx&*#&x&+xxxx&x+
++x&x+x++x&#&&+#&x&,+&#x+x+x++x-
+&&x+x+xx#+-,x+&#x--,-+x&xx+xx++
++xxxx&&&..,,&xx#x-,+.+xxx&+x+x+
x&x&&&++*, ,,&x-+--,, .+xx+xx+&+
---++x+#+-.,,xx+x&-,,,,xxx+x+x&+
-++&xx&&+-,,,+xx+&-,,,,-&+&++++x
--++&+x&+-,,,+xxxxx,,,-.=&xx&x#+
--&&xx+&x, ,,+x&+#+,,,+,#x+-xxx+
+&x+x+x&+,-,,x&-#&,,,-,x&&&xx#x+
--x.x++&--.,,&&-&#,,.-,&x+++&+&.
-&+x-++&.--,.##.&-,,-,+&x+x&-++-
-+x+x.-xx-,..xx+*x.,.,+&x&+xx+-.
 -&&x-#x&-  .+xx&.-+,+x#x+&+x++-
.,++x&&#&+++,xx+&+.x-x&&&x&xx+--
.+++x&&#&x&x-&++x&.xxx+xxx+++x-
.,+&#x+x&&&&&----x&-#x&&x-xxx+-,
-.+-x&xxx+++x+x&&&+#+++x&x+&xx..
.+-+&&xx*++&#+-+&&++#xx+x+&&&+-.
---x&x&x&&-++x&.^++xx&&&x.+x&+ .
  --x#xx+x&x-x++  &x&xxxx++x+x-,
,---+#&x++x#&xxxx&#&xxxx-++x-.,.
 -.&x&&x&+.+xx&x*&x+xx&-++&x.-+.
.++--&+xx+xxx+---+xxx&xxxx+x+-.+
----+xx&xx+x&#&&&&#x#xx++x---.+-
--.---+&&++-.xxx+x+xx-&&+--++x+&
-+---++-xx-+---.+--.---+----+---
ksize,pi = 8,3.14159
COSINE MATRICES: (normal & transpose)
 0.3536, 0.3536, 0.3536, 0.3536, 0.3536, 0.3536, 0.3536, 0.3536,
 0.4904, 0.4157, 0.2778, 0.0975, -0.0975, -0.2778, -0.4157, -0.4904,
 0.4619, 0.1913, -0.1913, -0.4619, -0.4619, -0.1913, 0.1913, 0.4619,
 0.4157, -0.0975, -0.4904, -0.2778, 0.2778, 0.4904, 0.0975, -0.4157,
 0.3536, -0.3536, -0.3536, 0.3536, 0.3536, -0.3536, -0.3536, 0.3536,
 0.2778, -0.4904, 0.0975, 0.4157, -0.4157, -0.0975, 0.4904, -0.2778,
 0.1913, -0.4619, 0.4619, -0.1913, -0.1913, 0.4619, -0.4619, 0.1913,
 0.0975, -0.2778, 0.4157, -0.4904, 0.4904, -0.4157, 0.2778, -0.0975,

 0.3536, 0.4904, 0.4619, 0.4157, 0.3536, 0.2778, 0.1913, 0.0975,
 0.3536, 0.4157, 0.1913, -0.0975, -0.3536, -0.4904, -0.4619, -0.2778,
 0.3536, 0.2778, -0.1913, -0.4904, -0.3536, 0.0975, 0.4619, 0.4157,
 0.3536, 0.0975, -0.4619, -0.2778, 0.3536, 0.4157, -0.1913, -0.4904,
```

```
 0.3536, -0.0975, -0.4619, 0.2778, 0.3536, -0.4157, -0.1913, 0.4904,
 0.3536, -0.2778, -0.1913, 0.4904, -0.3536, -0.0975, 0.4619, -0.4157,
 0.3536, -0.4157, 0.1913, 0.0975, -0.3536, 0.4904, -0.4619, 0.2778,
 0.3536, -0.4904, 0.4619, -0.4157, 0.3536, -0.2778, 0.1913, -0.0975,

----------------------BLOCK BEFORE & AFTER DCT below
 49.0000, 26.0000, 47.0000, 32.0000, 47.0000, 52.0000, 51.0000, 42.0000,
 46.0000, 32.0000, 43.0000, 34.0000, 50.0000, 35.0000, 33.0000, 19.0000,
 38.0000, 52.0000, 43.0000, 42.0000, 49.0000, 56.0000, 54.0000, 28.0000,
 28.0000, 49.0000, 56.0000, 43.0000, 59.0000, 50.0000, 52.0000, 47.0000,
 33.0000, 28.0000, 55.0000, 31.0000, 52.0000, 73.0000, 62.0000, 52.0000,
 19.0000, 23.0000, 37.0000, 35.0000, 42.0000, 73.0000, 55.0000, 53.0000,
 19.0000, 52.0000, 37.0000, 28.0000, 63.0000, 60.0000, 36.0000, 78.0000,
 23.0000, 40.0000, 30.0000, 20.0000, 28.0000, 65.0000, 27.0000, 50.0000,

 345.3750, -45.4638, -14.5339, 17.9775, -25.6250, -21.7799, 20.7677, 7.3691,
 1.4035, 36.3148, -8.6855, 9.0842, 6.3151, 30.5965, -11.8953, 22.9652,
 -25.3283, 10.1803, 14.2238, 4.1021, 11.8839, -9.2583, 7.3310, -7.5229,
 7.9154, -26.2988, 2.0203, 2.9490, 4.6372, 10.5871, 14.5873, 8.3466,
 4.3750, -3.4053, 5.8707, -1.8310, -6.1250, 6.1537, 11.2335, 3.7743,
 20.7993, -7.7024, 1.5719, -13.2223, 5.5466, -17.8620, -10.7962, 0.1783,
 -3.9857, -7.8429, 1.3310, 9.5470, -13.4463, 16.9022, 2.0262, -12.7968,
 14.1492, -4.8442, 5.6812, 5.4264, -3.8780, -4.0601, 1.4185, 1.0981,

----------------------
====================== quant matrix & stored block below
====================== DC scale = 6
 2, 4, 6, 8, 18, 16, 24, 26,
 1, 16, 18, 24, 32, 46, 54, 64,
 12, 24, 30, 46, 60, 66, 84, 94,
 12, 32, 40, 64, 78, 96, 106, 122,
 16, 32, 58, 80, 92, 116, 132, 156,
 24, 46, 66, 96, 116, 136, 160, 184,
 26, 52, 84, 104, 138, 162, 188, 220,
 32, 64, 92, 120, 158, 190, 220, 256,
 57, -45, -15, 18, -26, -22, 21, 7,
 1, 36, -9, 9, 6, 31, -12, 23,
 -25, 10, 14, 4, 12, -9, 7, -8,
 8, -26, 2, 3, 5, 11, 15, 8,
 4, -3, 6, -2, -6, 6, 11, 4,
 21, -8, 2, -13, 6, -18, -11, 0,
 -4, -8, 1, 10, -13, 17, 2, -13,
 14, -5, 6, 5, -4, -4, 1, 1,
====================== qmatrix = 2
====================== quant matrix & retrieved block below
====================== DC scale = 6
 2, 4, 6, 8, 18, 16, 24, 26,
 1, 16, 18, 24, 32, 46, 54, 64,
 12, 24, 30, 46, 60, 66, 84, 94,
 12, 32, 40, 64, 78, 96, 106, 122,
 16, 32, 58, 80, 92, 116, 132, 156,
 24, 46, 66, 96, 116, 136, 160, 184,
 26, 52, 84, 104, 138, 162, 188, 220,
 32, 64, 92, 120, 158, 190, 220, 256,
 336, -44, -12, 16, -18, -16, 0, 0,
 1, 32, 0, 0, 0, 0, 0, 0,
 -24, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0,
====================== qmatrix = 2
----------------------BLOCK BEFORE & AFTER INVERSE DCT below
 336.0000, -44.0000, -12.0000, 16.0000, -18.0000, -16.0000, 0.0000, 0.0000,
 1.0000, 32.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 -24.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,

 34.8910, 41.9708, 38.0274, 34.0537, 41.8734, 44.6035, 37.4130, 33.1967,
```

```
 35.9891, 43.2473, 39.6335, 36.0903, 44.3761, 47.5369, 40.6759, 36.6380,
 37.0228, 44.6105, 41.6057, 38.8581, 48.0051, 51.9615, 45.7095, 42.0012,
 36.4267, 44.4450, 42.2359, 40.5279, 50.8001, 55.7961, 50.3397, 47.0620,
 33.2963, 41.7807, 40.4327, 39.8500, 51.3401, 57.4613, 52.8661, 50.0544,
 28.1080, 37.0230, 36.4707, 36.9275, 49.5429, 56.7036, 52.9041, 50.5230,
 22.6473, 31.8919, 31.9485, 33.2010, 46.6775, 54.6339, 51.4434, 49.3919,
 19.1532, 28.5762, 28.9624, 30.6455, 44.5881, 52.9751, 50.1141, 48.2410,

----------------------
BREAK BEFORE ITERATION 1 (rogertst)!
 2, 6, 6, 12, 16, 24, 22, 30,
 2, 14, 20, 28, 36, 48, 48, 62,
 10, 18, 28, 40, 56, 64, 78, 92,
 10, 26, 48, 64, 76, 88, 110, 126,
 16, 32, 52, 72, 92, 114, 138, 156,
 22, 46, 66, 96, 114, 144, 164, 190,
 20, 48, 78, 106, 138, 166, 196, 224,
 26, 56, 94, 120, 160, 190, 216, 254,
(0 above)
 2, 8, 12, 16, 14, 18, 26, 28,
 8, 10, 24, 24, 32, 40, 52, 56,
 10, 18, 32, 48, 58, 72, 78, 96,
 12, 30, 42, 56, 78, 88, 106, 122,
 20, 38, 54, 72, 98, 112, 138, 160,
 22, 44, 72, 88, 116, 140, 164, 190,
 26, 56, 78, 112, 136, 166, 190, 216,
 28, 58, 90, 128, 154, 186, 218, 248,
(1 above)
 2, 4, 6, 8, 18, 16, 24, 26,
 1, 16, 18, 24, 32, 46, 54, 64,
 12, 24, 30, 46, 60, 66, 84, 94,
 12, 32, 40, 64, 78, 96, 106, 122,
 16, 32, 58, 80, 92, 116, 132, 156,
 24, 46, 66, 96, 116, 136, 160, 184,
 26, 52, 84, 104, 138, 162, 188, 220,
 32, 64, 92, 120, 158, 190, 220, 256,
(2 above)
 2, 8, 12, 10, 12, 24, 24, 24,
 4, 12, 18, 32, 34, 42, 48, 60,
 6, 18, 32, 42, 56, 72, 78, 96,
 14, 26, 42, 58, 80, 94, 104, 128,
 12, 32, 52, 76, 94, 112, 132, 158,
 22, 44, 72, 88, 112, 140, 166, 190,
 28, 54, 84, 110, 134, 168, 196, 220,
 26, 58, 92, 124, 154, 190, 216, 252,
(3 above)
 1, 6, 12, 12, 12, 22, 24, 24,
 6, 8, 20, 30, 40, 40, 56, 64,
 4, 24, 32, 46, 60, 64, 84, 88,
 16, 28, 44, 64, 72, 90, 104, 128,
 18, 36, 56, 74, 92, 112, 134, 156,
 24, 44, 66, 94, 120, 144, 168, 190,
 28, 48, 82, 106, 138, 162, 194, 220,
 32, 62, 96, 122, 156, 190, 218, 250,
(4 above)
 4, 8, 10, 14, 14, 20, 28, 32,
 1, 12, 20, 24, 32, 44, 48, 56,
 10, 18, 28, 48, 52, 64, 84, 94,
 12, 30, 40, 64, 80, 92, 112, 120,
 20, 38, 58, 74, 100, 120, 136, 154,
 18, 46, 70, 94, 112, 136, 164, 188,
 24, 54, 80, 112, 140, 164, 190, 222,
 28, 60, 88, 124, 156, 186, 220, 250,
(5 above)
 1, 6, 12, 10, 18, 24, 28, 32,
 8, 10, 16, 24, 36, 40, 56, 64,
 4, 22, 34, 40, 56, 68, 82, 90,
 10, 32, 48, 64, 80, 88, 106, 120,
 20, 38, 58, 76, 94, 118, 138, 154,
 22, 44, 66, 94, 116, 138, 164, 190,
 24, 50, 82, 104, 132, 162, 196, 222,
 24, 60, 92, 120, 158, 192, 218, 250,
(6 above)
 4, 1, 10, 8, 16, 22, 24, 30,
```

```
 8, 10, 18, 24, 40, 44, 48, 56,
 4, 18, 32, 42, 58, 68, 76, 96,
 14, 30, 40, 62, 76, 96, 106, 122,
 16, 34, 60, 72, 92, 112, 136, 160,
 22, 40, 64, 92, 116, 144, 164, 190,
 24, 54, 76, 110, 132, 164, 194, 224,
 32, 58, 88, 122, 152, 190, 224, 250,
(7 above)
---------------------BLOCK BEFORE & AFTER DCT below
 49.0000, 26.0000, 47.0000, 32.0000, 47.0000, 52.0000, 51.0000, 42.0000,
 46.0000, 32.0000, 43.0000, 34.0000, 50.0000, 35.0000, 33.0000, 19.0000,
 38.0000, 52.0000, 43.0000, 42.0000, 49.0000, 56.0000, 54.0000, 28.0000,
 28.0000, 49.0000, 56.0000, 43.0000, 59.0000, 50.0000, 52.0000, 47.0000,
 33.0000, 28.0000, 55.0000, 31.0000, 52.0000, 73.0000, 62.0000, 52.0000,
 19.0000, 23.0000, 37.0000, 35.0000, 42.0000, 73.0000, 55.0000, 53.0000,
 19.0000, 52.0000, 37.0000, 28.0000, 63.0000, 60.0000, 36.0000, 78.0000,
 23.0000, 40.0000, 30.0000, 20.0000, 28.0000, 65.0000, 27.0000, 50.0000,

 345.3750, -45.4638, -14.5339, 17.9775, -25.6250, -21.7799, 20.7677, 7.3691,
 1.4035, 36.3148, -8.6855, 9.0842, 6.3151, 30.5965, -11.8953, 22.9652,
 -25.3283, 10.1803, 14.2238, 4.1021, 11.8839, -9.2583, 7.3310, -7.5229,
 7.9154, -26.2988, 2.0203, 2.9490, 4.6372, 10.5871, 14.5873, 8.3466,
 4.3750, -3.4053, 5.8707, -1.8310, -6.1250, 6.1537, 11.2335, 3.7743,
 20.7993, -7.7024, 1.5719, -13.2223, 5.5466, -17.8620, -10.7962, 0.1783,
 -3.9857, -7.8429, 1.3310, 9.5470, -13.4463, 16.9022, 2.0262, -12.7968,
 14.1492, -4.8442, 5.6812, 5.4264, -3.8780, -4.0601, 1.4185, 1.0981,

----------------------
====================== quant matrix & stored block below
====================== DC scale = 6
 2, 4, 6, 8, 18, 16, 24, 26,
 1, 16, 18, 24, 32, 46, 54, 64,
 12, 24, 30, 46, 60, 66, 84, 94,
 12, 32, 40, 64, 78, 96, 106, 122,
 16, 32, 58, 80, 92, 116, 132, 156,
 24, 46, 66, 96, 116, 136, 160, 184,
 26, 52, 84, 104, 138, 162, 188, 220,
 32, 64, 92, 120, 158, 190, 220, 256,
 57, -45, -15, 18, -26, -22, 21, 7,
 1, 36, -9, 9, 6, 31, -12, 23,
 -25, 10, 14, 4, 12, -9, 7, -8,
 8, -26, 2, 3, 5, 11, 15, 8,
 4, -3, 6, -2, -6, 6, 11, 4,
 21, -8, 2, -13, 6, -18, -11, 0,
 -4, -8, 1, 10, -13, 17, 2, -13,
 14, -5, 6, 5, -4, -4, 1, 1,
====================== qmatrix = 2
====================== quant matrix & retrieved block below
====================== DC scale = 6
 2, 4, 6, 8, 18, 16, 24, 26,
 1, 16, 18, 24, 32, 46, 54, 64,
 12, 24, 30, 46, 60, 66, 84  94,
 12, 32, 40, 64, 78, 96, 106, 122,
 16, 32, 58, 80, 92, 116, 132, 156,
 24, 46, 66, 96, 116, 136, 160, 184,
 26, 52, 84, 104, 138, 162, 188, 220,
 32, 64, 92, 120, 158, 190, 220, 256,
 336, -44, -12, 16, -18, -16, 0, 0,
 1, 32, 0, 0, 0, 0, 0, 0,
 -24, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0,
====================== qmatrix = 2
---------------------BLOCK BEFORE & AFTER INVERSE DCT below
 336.0000, -44.0000, -12.0000, 16.0000, -18.0000, -16.0000, 0.0000, 0.0000,
 1.0000, 32.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 -24.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
```

```
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,

 34.8910, 41.9708, 38.0274, 34.0537, 41.8734, 44.6035, 37.4130, 33.1967,
 35.9891, 43.2473, 39.6335, 36.0903, 44.3761, 47.5369, 40.6759, 36.6380,
 37.0228, 44.6105, 41.6057, 38.8581, 48.0051, 51.9615, 45.7095, 42.0012,
 36.4267, 44.4450, 42.2359, 40.5279, 50.8001, 55.7961, 50.3397, 47.0620,
 33.2963, 41.7807, 40.4327, 39.8500, 51.3401, 57.4613, 52.8661, 50.0544,
 28.1080, 37.0230, 36.4707, 36.9275, 49.5429, 56.7036, 52.9041, 50.5230,
 22.6473, 31.8919, 31.9485, 33.2010, 46.6775, 54.6339, 51.4434, 49.3919,
 19.1532, 28.5762, 28.9624, 30.6455, 44.5881, 52.9751, 50.1141, 48.2410,

----------------------
   original_bpp = 8.00000
   compress_bpp = 2.81046
   compress_mse = 402.87479
   compress_rms = 20.07174
   compress_snr = 22.07910
   compress_rat = 0.12729
SUCCESSFUL RUN!
```

```
QUANMAIN (version 1.4):
Running GENETIC ALGORITHM method!
  bitsused=8
  ksize=8
  rows=256
  cols=256
  numquants=8
  maxclust=8
  initrad=0
  kohiters=1
  threshval=-1
  learnrate=0.0000
  adjrate=0.0000
  iters=5
  tsize=2
  psize,xover,pc,pm=4,2,10000,600
  origfile=mandrill.raw
  edgefile=mand_ga.edg
  compfile=mand_ga.cmp
  expfile=mand_ga.exp

The min, max and average of the gradients are:
  Min =         0
  Max =       255
  Avg =      83.8
  Thr =       237 (auto='.9*(Max-Avg)')
Edge detected file printed to file mand_ga.edg!
-+++x+xxx+x-#&x&#&xxx+--+xxx++-++
.++xx&x#&&x&xxx&+-#xxx&&x&#xxx+-
-&xxxx&x++-+++&&&*+x&xx+&#xx&+&+
+xxx&x&+&*-x#x-+x+==x+&+-+x&+xxx
++&+xx-+-&#&*&&xx&&#&&x+&xx++x#-
++xx+x+xx&+xx+#&xx&*#&x&+xxxx&x+
++x&x+x++x&#&&+#&x&,+&#x+x+x++x-
+&&x+x+xx#+-,x+&#x--,-+x&xx+xx++
++xxxx&&&..,,&xx#x-,+.+xxx&+x+x+
x&x&&&--*,-,,&x-+--,,-.+xx+xx+&+
-++++x+=--.,,xx+x&-,,,,xxx+x+x&+
-++&xx&&+-,,,-xx+&-,,,,-&-&++-+x
-+++&+x&+-,,,-xxxxx,,,-,#&xx&x=-
--&&xx+&x,-,,+x&+#+,,,+,#x+-xxx+
+&x+x+x&+,-,,x&-#&,,,-,x&&&xx#x+
--x.x++&--.,,&&-&#,,.-,&x+++&+&.
-&+x-++&.--,.##.&-,,-,+&x+x&-++-
-+x+x.-xx-,..xx+*x.,,,+&x&+xx+-
 -&&x-#x&-.-.+xx&.-+,+x#x+&+x++-
.,++x&&#&+++,xx+&+.x-x&&&x&xx+--
.+++x&&#&x&x-&++x&.xxx+xxx+++x-
.,+&#x+x&&&&&----x&-#x&&x-xxx+-,
-.+-x&xxx+++x+x&&&+#+++x&x+&xx..
.+-+&&xx*++x+--+&&++#xx+x+&&&+-.
---x&x&x&&-++ :&+++++xx+&&x.+x&+-,
  --x#xx+x&x-x++--&x&xxxx++x+x-,
.---+#&x++x#&xxxx&#&xxxx-++x-.,.
. .&x&&x&+.+xx&x*&x+xx&-++&x.-+.
.++--&+xx+xxx+---+xxx&xxxx+x+-.+
----+xx&xx+x&#&&&&#x#xx++x---.+-
-- ---+&&++-.xxx+x+xx-&&+--++x+&
-+---++-xx-+---.+--.---+----+---
ksize,pi = 8,3.14159
COSINE MATRICES: (normal & transpose)
 0.3536, 0.3536, 0.3536, 0.3536, 0.3536, 0.3536, 0.3536, 0.3536,
 0.4904, 0.4157, 0.2778, 0.0975, -0.0975, -0.2778, -0.4157, -0.4904,
 0.4619, 0.1913, -0.1913, -0.4619, -0.4619, -0.1913, 0.1913, 0.4619,
 0.4157, -0.0975, -0.4904, -0.2778, 0.2778, 0.4904, 0.0975, -0.4157,
 0.3536, -0.3536, -0.3536, 0.3536, 0.3536, -0.3536, -0.3536, 0.3536,
 0.2778, -0.4904, 0.0975, 0.4157, -0.4157, -0.0975, 0.4904, -0.2778,
 0.1913, -0.4619, 0.4619, -0.1913, -0.1913, 0.4619, -0.4619, 0.1913,
 0.0975, -0.2778, 0.4157, -0.4904, 0.4904, -0.4157, 0.2778, -0.0975,

 0.3536, 0.4904, 0.4619, 0.4157, 0.3536, 0.2778, 0.1913, 0.0975,
 0.3536, 0.4157, 0.1913, -0.0975, -0.3536, -0.4904, -0.4619, -0.2778,
 0.3536, 0.2778, -0.1913, -0.4904, -0.3536, 0.0975, 0.4619, 0.4157,
```

```
 0.3536, 0.0975, -0.4619, -0.2778, 0.3536, 0.4157, -0.1913, -0.4904,
 0.3536, -0.0975, -0.4619, 0.2778, 0.3536, -0.4157, -0.1913, 0.4904,
 0.3536, -0.2778, -0.1913, 0.4904, -0.3536, -0.0975, 0.4619, -0.4157,
 0.3536, -0.4157, 0.1913, 0.0975, -0.3536, 0.4904, -0.4619, 0.2778,
 0.3536, -0.4904, 0.4619, -0.4157, 0.3536, -0.2778, 0.1913, -0.0975,

--------------------BLOCK BEFORE & AFTER DCT below
 49.0000, 26.0000, 47.0000, 32.0000, 47.0000, 52.0000, 51.0000, 42.0000,
 45.0000, 32.0000, 43.0000, 34.0000, 50.0000, 35.0000, 35.0000, 19.0000,
 38.0000, 52.0000, 43.0000, 42.0000, 49.0000, 56.0000, 54.0000, 28.0000,
 28.0000, 49.0000, 56.0000, 43.0000, 59.0000, 50.0000, 52.0000, 47.0000,
 33.0000, 28.0000, 55.0000, 31.0000, 52.0000, 73.0000, 62.0000, 52.0000,
 19.0000, 23.0000, 37.0000, 35.0000, 42.0000, 73.0000, 55.0000, 53.0000,
 19.0000, 52.0000, 37.0000, 28.0000, 63.0000, 60.0000, 36.0000, 78.0000,
 25.0000, 40.0000, 30.0000, 20.0000, 28.0000, 65.0000, 27.0000, 50.0000,

 345.3750, -45.4638, -14.5339, 17.9775, -25.4250, -21.7799, 20.7677, 7.3691,
 1.4035, 36.3148, -8.6855, 9.0842, 6.3151, 30.5965, -11.8953, 22.9652,
 -25.3283, 10.1803, 14.2238, 4.1021, 11.8839, -9.2583, 7.3310, -7.5229,
 7.9154, -26.2988, 2.0205, 2.9490, 4.6372, 10.5871, 14.5873, 8.3466,
 4.3750, -3.4053, 5.8707, -1.8311, -6.1250, 6.1537, 11.2335, 3.7743,
 20.7993, -7.7024, 1.5719, -13.2223, 5.5466, -17.8620, -10.7962, 0.1783,
 -3.9857, -7.8429, 1.3310, 9.5470, -13.4483, 16.9022, 2.0262, -12.7968,
 14.1492, -4.8442, 5.6812, 5.4264, -5.8750, -4.0601, 1.4183, 1.0981,

----------------------
********************** quant matrix & stored block below
********************** DC scale = 8
 1, 1, 4, 8, 16, 18, 24, 24,
 6, 8, 20, 28, 40, 40, 52, 64,
 12, 20, 34, 42, 56, 66, 78, 88,
 8, 28, 44, 64, 78, 90, 106, 122,
 20, 38, 60, 80, 98, 120, 138, 158,
 18, 40, 70, 96, 114, 140, 162, 192,
 20, 50, 78, 110, 134, 164, 188, 218,
 30, 60, 88, 126, 156, 186, 216, 254,
 57, -45, -15, 18, -24, -22, 21, 7,
 1, 36, -9, 9, 6, 31, -12, 23,
 -25, 10, 14, 4, 12, -9, 7, -8,
 8, -26, 2, 3, 5, 11, 15, 8,
 4, -3, 6, -2, -6, 6, 11, 4,
 21, -8, 2, -13, 6, -18, -11, 3,
 -4, -8, 1, 10, -13, 17, 2, -13,
 14, -5, 6, 5, -4, -4, 1, 1,
********************** qmatrix = 1
********************** quant matrix & retrieved block below
********************** DC scale   8
 1, 1, 4, 8, 16, 18, 24, 24,
 6, 8, 20, 28, 40, 40, 52, 64,
 12, 20, 34, 42, 56, 66, 78, 88,
 8, 28, 44, 64, 78, 90, 106, 122,
 20, 38, 60, 80, 98, 120, 138, 158,
 18, 40, 70, 96, 114, 140, 162, 192,
 20, 50, 78, 110, 134, 164, 188, 218,
 30, 60, 88, 126, 156, 186, 216, 254,
 342, -45, -12, 16, -16, -18, 0, 0,
 0, 32, 0, 0, 0, 0, 0, 0,
 -24, 0, 0, 0, 0, 0, 0, 0,
 8, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0,
 18, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0,
********************** qmatrix = 2
--------------------BLOCK BEFORE & AFTER INVERSE DCT below
 342.0000, -45.0000, -12.0000, 16.0000, -16.0000, -18.0000, 0.0000, 0.0000,
 0.0000, 32.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 -24.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 8.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 18.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
```

38.3915, 45.4409, 41.1306, 37.4955, 45.9721, 49.0410, 40.4835, 37.3368,
33.0756, 40.4054, 36.4226, 33.2182, 42.1609, 44.8603, 37.4324, 34.4640,
36.7885, 44.4458, 41.0740, 38.6652, 48.4691, 51.7642, 45.1453, 42.5065,
38.8824, 46.9704, 44.3942, 43.0250, 53.9542, 55.3888, 52.4655, 50.2573,
32.1009, 40.6549, 38.9400, 38.6960, 50.8430, 56.3039, 51.3408, 49.5987,
29.6027, 38.5873, 37.6680, 38.4636, 51.7358, 58.2353, 54.0688, 52.7573,
26.8212, 36.1354, 35.8250, 37.4163, 51.5497, 58.8448, 55.2873, 54.3054,
17.0131, 26.5056, 26.5249, 28.5467, 43.1462, 50.8720, 47.6440, 46.8405,

---

----------BLOCK BEFORE & AFTER DCT below

49.0000, 26.0000, 47.0000, 32.0000, 47.0000, 52.0000, 51.0000, 42.0000,
46.0000, 32.0000, 43.0000, 34.0000, 50.0000, 35.0000, 33.0000, 19.0000,
38.0000, 52.0000, 43.0000, 42.0000, 49.0000, 56.0000, 54.0000, 28.0000,
28.0000, 49.0000, 56.0000, 43.0000, 59.0000, 50.0000, 52.0000, 47.0000,
33.0000, 28.0000, 55.0000, 31.0000, 52.0000, 73.0000, 62.0000, 52.0000,
19.0000, 23.0000, 37.0000, 35.0000, 42.0000, 73.0000, 55.0000, 53.0000,
19.0000, 52.0000, 37.0000, 28.0000, 63.0000, 60.0000, 36.0000, 78.0000,
23.0000, 40.0000, 30.0000, 20.0000, 28.0000, 65.0000, 27.0000, 50.0000,

345.3750, -45.4638, -14.5339, 17.9775, -25.6250, -21.7799, 20.7677, 7.3691,
1.4035, 36.3148, -8.6855, 9.0842, 6.3151, 30.5965, -11.8953, 22.9652,
-25.3283, 10.1803, 14.2238, 4.1021, 11.8839, -9.2583, 7.3310, -7.5229,
7.9154, -26.2988, 2.0203, 2.9490, 4.6372, 10.5871, 14.5873, 8.3466,
4.3750, -3.4053, 5.8707, -1.8310, -6.1250, 6.1537, 11.2335, 3.7743,
20.7993, -7.7024, 1.5719, -13.2223, 5.5466, -17.8620, -10.7962, 0.1783,
-3.9857, -7.8429, 1.3310, 9.5470, -13.4463, 16.9023, 2.0262, -12.7968,
14.1492, -4.8442, 5.6812, 5.4264, -3.8780, -4.0601, 1.4185, 1.0981,

---

========== quant matrix & stored block below
========== DC scale = 6
2, 4, 12, 10, 20, 20, 20, 30,
6, 10, 20, 26, 36, 48, 52, 56,
6, 16, 36, 46, 58, 64, 80, 90,
8, 24, 44, 56, 78, 96, 110, 122,
20, 36, 60, 76, 92, 116, 136, 158,
22, 46, 72, 92, 120, 136, 166, 184,
28, 52, 78, 106, 132, 162, 196, 218,
24, 64, 94, 128, 152, 188, 220, 256,
57, -45, -15, 18, -26, -22, 21, 7,
1, 36, -9, 9, 6, 31, -12, 23,
-25, 10, 14, 4, 12, -9, 7, -8,
8, -26, 2, 3, 5, 11, 15, 8,
4, -3, 6, -2, -6, 6, 11, 4,
21, -8, 2, -13, 6, -18, -11, 0,
-4, -8, 1, 10, -13, 17, 2, -13,
14, -5, 6, 5, -4, -4, 1, 1,
========== qmatrix = 2
========== quant matrix & retrieved block below
========== DC scale = 6
2, 4, 12, 10, 20, 20, 20, 30,
6, 10, 20, 26, 36, 48, 52, 56,
6, 16, 36, 46, 58, 64, 80, 90,
8, 24, 44, 56, 78, 96, 110, 122,
20, 36, 60, 76, 92, 116, 136, 158,
22, 46, 72, 92, 120, 136, 166, 184,
28, 52, 78, 106, 132, 162, 196, 218,
24, 64, 94, 128, 152, 188, 220, 256,
336, -44, -12, 10, -20, -20, 20, 0,
0, 30, 0, 0, 0, 0, 0, 0,
-24, 0, 0, 0, 0, 0, 0, 0,
8, -24, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
========== qmatrix = 2
----------BLOCK BEFORE & AFTER INVERSE DCT below
336.0000, -44.0000, -12.0000, 10.0000, -20.0000, -20.0000, 20.0000, 0.0000,
0.0000, 30.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
-24.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
8.0000, -24.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,

```
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,

 30.3478, 36.3015, 40.4046, 32.3656, 42.3405, 51.2642, 39.0544, 41.9509,
 36.1348, 41.3361, 44.0487, 34.2131, 42.2017, 49.3087, 35.7086, 37.8525,
 40.8649, 45.6713, 47.6542, 36.8651, 43.8217, 49.9753, 35.6454, 37.3945,
 38.6085, 44.1994, 47.6322, 38.7374, 47.7443, 55.7922, 42.9122, 45.4458,
 30.7710, 37.7944, 43.8739, 38.4372, 51.1873, 62.6933, 52.4600, 56.4261,
 23.9223, 31.7303, 39.2596, 35.7173, 50.5177, 63.9180, 55.1345, 59.8852,
 22.1581, 29.5712, 36.3708, 31.8750, 45.6434, 58.0902, 48.5770, 52.9328,
 23.3529, 30.0135, 35.4227, 29.1102, 40.9123, 51.5426, 40.6390, 44.2423,

----------------------
----------------------BLOCK BEFORE & AFTER DCT below
 49.0000, 26.0000, 47.0000, 32.0000, 47.0000, 52.0000, 51.0000, 42.0000,
 46.0000, 32.0000, 43.0000, 34.0000, 50.0000, 35.0000, 33.0000, 19.0000,
 38.0000, 52.0000, 43.0000, 42.0000, 49.0000, 56.0000, 54.0000, 28.0000,
 28.0000, 49.0000, 56.0000, 43.0000, 59.0000, 50.0000, 52.0000, 47.0000,
 33.0000, 28.0000, 55.0000, 31.0000, 52.0000, 73.0000, 62.0000, 52.0000,
 19.0000, 23.0000, 37.0000, 35.0000, 42.0000, 73.0000, 55.0000, 53.0000,
 19.0000, 52.0000, 37.0000, 28.0000, 63.0000, 60.0000, 36.0000, 78.0000,
 23.0000, 40.0000, 30.0000, 20.0000, 28.0000, 65.0000, 27.0000, 50.0000,

 345.3750, -45.4638, -14.5339, 17.9775, -25.6250, -21.7799, 20.7677, 7.3691,
 1.4035, 36.3148, -8.6855, 9.0842, 6.3151, 30.5965, -11.8953, 22.9652,
 -25.3283, 10.1803, 14.2238, 4.1021, 11.8839, -9.2583, 7.3310, -7.5229,
 7.9154, -26.2988, 2.0203, 2.9490, 4.6372, 10.5871, 14.5873, 8.3466,
 4.3750, -3.4053, 5.8707, -1.8310, -6.1250, 6.1537, 11.2335, 3.7743,
 20.7993, -7.7024, 1.5719, -13.2223, 5.5466, -17.8620, -10.7962, 0.1783,
 -3.9857, -7.8429, 1.3310, 9.5470, -13.4463, 16.9022, 2.0262, -12.7968,
 14.1492, -4.8442, 5.6812, 5.4264, -3.8780, -4.0601, 1.4185, 1.0981,

----------------------
======================== quant matrix & stored block below
======================== DC scale = 6
 1, 6, 4, 16, 14, 20, 28, 24,
 8, 12, 22, 26, 32, 42, 48, 64,
 10, 16, 32, 42, 56, 64, 84, 92,
 8, 26, 40, 58, 74, 96, 108, 128,
 14, 32, 60, 74, 98, 120, 136, 160,
 24, 40, 72, 92, 114, 144, 166, 188,
 24, 48, 78, 106, 136, 168, 188, 218,
 26, 56, 94, 120, 152, 190, 222, 254,
 57, -45, -15, 18, -26, -22, 21, 7,
 1, 36, -9, 9, 6, 31, -12, 23,
 -25, 10, 14, 4, 12, -9, 7, -8,
 8, -26, 2, 3, 5, 11, 15, 8,
 4, -3, 6, -2, -6, 6, 11, 4,
 21, -8, 2, -13, 6, -18, -11, 0,
 -4, -8, 1, 10, -13, 17, 2, -13,
 14, -5, 6, 5, -4, -4, _  1,
======================== _atrix - 2
======================== quant matrix & retrieved block below
======================== DC scale = 6
 1, 6, 4, 16, 14, 20, 28, 24,
 8, 12, 22, 26, 32, 42, 48, 64,
 10, 16, 32, 42, 56, 64, 84, 92,
 8, 26, 40, 58, 74, 96, 108, 128,
 14, 32, 60, 74, 98, 120, 136, 160,
 24, 40, 72, 92, 114, 144, 166, 188,
 24, 48, 78, 106, 136, 168, 188, 218,
 26, 56, 94, 120, 152, 190, 222, 254,
 342, -42, -12, 16, -14, -20, 0, 0,
 0, 36, 0, 0, 0, 0, 0, 0,
 -20, 0, 0, 0, 0, 0, 0, 0,
 8, -26, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0,
======================== qmatrix = 2
----------------------BLOCK BEFORE & AFTER INVERSE DCT below
 342.0000, -42.0000, -12.0000, 16.0000, -14.0000, -20.0000, 0.0000, 0.0000,
```

```
0.0000, 36.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
-20.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
8.0000, -26.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,

33.4119, 41.1858, 37.5340, 35.5775, 46.1612, 48.9085, 42.0095, 40.4873,
39.1000, 46.0782, 40.9561, 37.0787, 45.5832, 46.4097, 38.0404, 35.7226,
43.2683, 49.8548, 44.0088, 39.1854, 46.6661, 46.5467, 37.4535, 34.7439,
39.8903, 47.3739, 43.1855, 40.5281, 50.3531, 52.3995, 44.9640, 43.1514,
30.9339, 40.0202, 38.7934, 40.0053, 54.0186, 59.9344, 55.4604, 55.2506,
23.7291, 33.7125, 34.1434, 37.5211, 53.8786, 61.9603, 59.1440, 59.8313,
22.4855, 32.0771, 31.7840, 34.2159, 49.5496, 56.6854, 53.1451, 53.4407,
24.3467, 33.1427, 31.3793, 31.8903, 45.1448, 50.3597, 45.3492, 44.8491,

----------------------
----------------------BLOCK BEFORE & AFTER DCT below
 49.0000, 26.0000, 47.0000, 32.0000, 47.0000, 52.0000, 51.0000, 42.0000,
 46.0000, 32.0000, 43.0000, 34.0000, 50.0000, 35.0000, 33.0000, 19.0000,
 38.0000, 52.0000, 43.0000, 42.0000, 49.0000, 56.0000, 54.0000, 28.0000,
 28.0000, 49.0000, 56.0000, 43.0000, 59.0000, 50.0000, 52.0000, 47.0000,
 33.0000, 28.0000, 55.0000, 31.0000, 52.0000, 73.0000, 62.0000, 52.0000,
 19.0000, 23.0000, 37.0000, 35.0000, 42.0000, 73.0000, 55.0000, 53.0000,
 19.0000, 52.0000, 37.0000, 28.0000, 63.0000, 60.0000, 36.0000, 78.0000,
 23.0000, 40.0000, 30.0000, 20.0000, 28.0000, 65.0000, 27.0000, 50.0000,

 345.3750, -45.4638, -14.5339, 17.9775, -25.6250, -21.7799, 20.7677, 7.3691,
 1.4035, 36.3148, -8.6855, 9.0842, 6.3151, 30.5965, -11.8953, 22.9652,
 -25.3283, 10.1803, -14.2238, 4.1021, 11.8839, -9.2583, 7.3310, -7.5229,
 7.9154, -26.2988, 2.0203, 2.9490, 4.6372, -10.5871, -14.5873, 8.3466,
 4.3750, -3.4053, 5.8707, -1.8310, -6.1250, 6.1537, 11.2335, 3.7743,
 20.7993, -7.7024, 1.5719, -13.2223, 5.5466, -17.8620, -10.7962, 0.1783,
 -3.9857, -7.8429, 1.3310, 9.5470, -13.4463, 16.9022, 2.0262, -12.7968,
 14.1492, -4.8442, 5.6812, 5.4264, -3.8780, -4.0601, 1.4185, 1.0981,

----------------------
========================= quant matrix & stored block below
========================= DC scale = 6
 2, 8, 4, 10, 14, 18, 28, 30,
 6, 14, 20, 24, 38, 46, 52, 56,
 4, 16, 34, 48, 58, 70, 84, 96,
 8, 30, 44, 64, 76, 96, 106, 120,
 16, 40, 52, 80, 92, 114, 138, 154,
 16, 46, 70, 92, 118, 142, 160, 190,
 28, 48, 82, 108, 136, 160, 190, 224,
 28, 62, 96, 120, 160, 188, 224, 256,
 57, -45, -15, 18, -26, -22, 21, 7,
 1, 36, -9, 9, 6, 31, -12, 23,
 -25, 10, 14, 4, 12, -9, 7, -8,
 8, -26, 2, -3, 5, 11, -15, 5,
 4, -3, 6, -8, -6, 6, 11, 4,
 21, -8, 2, -13, 6, -18, -11, 0,
 -4, -8, 1, 10, -13, 17, 2, -13,
 14, -5, 6, 5, -4, -4, 1, 1,
========================= qmatrix   2
========================= quant matrix & retrieved block below
========================= DC scale   6
 2, 8, 4, 10, 14, 18, 28, 30,
 6, 14, 20, 24, 38, 46, 52, 56,
 4, 16, 34, 48, 58, 70, 84, 96,
 8, 30, 44, 64, 76, 96, 106, 120,
 16, 40, 52, 80, 92, 114, 138, 154,
 16, 46, 70, 92, 118, 142, 160, 190,
 28, 48, 82, 108, 136, 160, 190, 224,
 28, 62, 96, 120, 160, 188, 224, 256,
 336, -40, -12, 10, -14, -18, 0, 0,
 0, 28, 0, 0, 0, 0, 0, 0,
 -24, 0, 0, 0, 0, 0, 0, 0,
 8, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0,
 16, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0,
```

```
 0, 0, 0, 0, 0, 0, 0, 0,
********************* matrix = 2
----------------------BLOCK BEFORE & AFTER INVERSE DCT below
 336.0000, -40.0000, -12.0000, 10.0000, -24.0000, -16.0000, 0.0000, 0.0000
 0.0000, 28.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 -24.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 8.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 16.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,

 36.6181, 44.3708, 40.9206, 37.3695, 44.7054, 45.8582, 39.1607, 37.6173,
 32.0919, 40.0006, 36.8388, 33.6644, 41.4081, 42.9377, 36.5287, 35.1413,
 35.6596, 43.8567, 41.2277, 38.7496, 47.2468, 49.4726, 43.5964, 42.4974,
 37.8821, 46.4560, 44.5232, 42.9547, 52.4365, 55.5719, 50.3919, 49.6697,
 32.0712, 41.0529, 39.8737, 39.2897, 49.8372, 53.9572, 49.5307, 49.2164,
 29.7016, 39.0600, 38.5770, 38.9026, 50.4547, 55.4643, 51.7340, 51.7964,
 26.7749, 36.4218, 36.4716, 37.4934, 49.7790, 55.5048, 52.3074, 52.6582,
 17.6565, 27.4594, 27.7975, 29.1962, 41.8896, 47.9922, 45.0831, 45.5899,

----------------------
BREAK BEFORE ITERATION 1 (rogertst)!
 4, 4, 8, 10, 12, 24, 20, 24,
 1, 16, 16, 24, 34, 40, 56, 62,
 10, 22, 36, 48, 56, 64, 78, 90,
 10, 24, 40, 56, 72, 96, 112, 122,
 16, 36, 52, 78, 100, 112, 136, 154,
 22, 48, 70, 90, 116, 136, 164, 190,
 22, 50, 76, 110, 132, 164, 190, 216,
 24, 64, 92, 120, 154, 190, 216, 254,
rat_best,quant_bestiter = [0.16191,0]
(0 above)
 2, 2, 10, 8, 16, 24, 26, 32,
 2, 12, 16, 32, 32, 40, 48, 56,
 8, 16, 32, 40, 58, 72, 76, 92,
 8, 28, 44, 62, 76, 88, 104, 128,
 14, 34, 56, 72, 100, 120, 138, 152,
 18, 44, 72, 96, 118, 144, 162, 184,
 22, 52, 78, 110, 136, 166, 190, 220,
 28, 62, 92, 124, 154, 186, 222, 254,
rat_best,quant_bestiter = [0.12283,0]
(1 above)
 2, 8, 4, 10, 14, 18, 28, 30,
 6, 14, 20, 24, 38, 46, 52, 56,
 4, 16, 34, 48, 58, 70, 84, 96,
 8, 30, 44, 64, 76, 96, 106, 120,
 16, 40, 52, 80, 92, 114, 138, 154,
 16, 46, 70, 92, 118, 142, 160, 190,
 28, 48, 82, 108, 136, 160, 190, 224,
 28, 62, 96, 120, 160, 188, 224, 256,
rat_best,quant_bestiter   [0.12308,0]
(2 above)
 2, 4, 4, 10, 18, 18, 28, 28,
 4, 16, 18, 26, 38, 42, 52, 64,
 4, 20, 32, 40, 52, 64, 80, 96,
 8, 24, 48, 56, 76, 94, 110, 126,
 18, 32, 54, 78, 100, 118, 134, 158,
 16, 46, 64, 96, 120, 142, 168, 190,
 28, 56, 76, 106, 134, 162, 194, 216,
 28, 56, 96, 120, 160, 192, 224, 256,
rat_best,quant_bestiter = [0.12073,0]
(3 above)
 2, 2, 10, 10, 18, 22, 26, 32,
 2, 12, 18, 28, 34, 42, 48, 56,
 6, 20, 28, 44, 58, 70, 76, 94,
 14, 26, 42, 64, 80, 92, 108, 128,
 20, 40, 56, 78, 98, 120, 136, 158,
 20, 40, 68, 88, 112, 136, 164, 192,
 28, 56, 80, 104, 136, 160, 192, 222,
 24, 62, 94, 124, 152, 184, 222, 256,
rat_best,quant_bestiter = [0.12382,0]
(4 above)
 4, 4, 6, 16, 14, 22, 20, 30,
```

```
 2, 8, 22, 28, 38, 44, 54, 60,
 6, 18, 34, 48, 64, 68, 82, 96,
 8, 24, 42, 56, 72, 90, 106, 124,
 14, 40, 58, 76, 100, 116, 138, 154,
 24, 48, 66, 94, 112, 144, 164, 186,
 28, 54, 76, 104, 138, 162, 194, 216,
 32, 60, 88, 128, 156, 188, 224, 250,
rat_best,quant_bestiter = [0.12805,0]
(5 above)
 2, 8, 4, 14, 12, 20, 22, 28,
 1, 10, 24, 32, 34, 44, 54, 62,
 4, 22, 34, 48, 56, 68, 78, 96,
 14, 30, 42, 58, 76, 92, 104, 122,
 20, 32, 58, 74, 100, 114, 136, 160,
 22, 40, 64, 94, 114, 142, 168, 184,
 28, 52, 80, 106, 140, 166, 194, 220,
 28, 60, 90, 126, 154, 190, 220, 254,
rat_best,quant_bestiter = [0.12412,0]
(6 above)
 2, 2, 8, 16, 12, 18, 24, 28,
 4, 8, 22, 30, 40, 44, 50, 58,
 8, 16, 30, 40, 56, 64, 80, 92,
 14, 28, 40, 64, 72, 92, 108, 120,
 20, 38, 52, 76, 92, 114, 140, 152,
 22, 40, 70, 88, 116, 138, 168, 188,
 22, 56, 76, 108, 136, 162, 196, 218,
 26, 62, 90, 122, 158, 190, 218, 248,
rat_best,quant_bestiter = [0.12713,0]
(7 above)
----------------------BLOCK BEFORE & AFTER DCT below
 49.0000, 26.0000, 47.0000, 32.0000, 47.0000, 52.0000, 51.0000, 42.0000,
 46.0000, 32.0000, 43.0000, 34.0000, 50.0000, 35.0000, 33.0000, 19.0000,
 38.0000, 52.0000, 43.0000, 42.0000, 49.0000, 56.0000, 54.0000, 28.0000,
 28.0000, 49.0000, 56.0000, 43.0000, 59.0000, 50.0000, 52.0000, 47.0000,
 33.0000, 28.0000, 55.0000, 31.0000, 52.0000, 73.0000, 62.0000, 52.0000,
 19.0000, 23.0000, 37.0000, 35.0000, 42.0000, 73.0000, 55.0000, 53.0000,
 19.0000, 52.0000, 37.0000, 28.0000, 63.0000, 60.0000, 36.0000, 78.0000,
 23.0000, 40.0000, 30.0000, 20.0000, 28.0000, 65.0000, 27.0000, 50.0000,

 345.3750, -45.4638, -14.5339, 17.9775, -25.6250, -21.7799, 20.7677, 7.3691
 1.4035, 36.3148, -8.6855, 9.0842, 6.3151, 30.5965, -11.8953, 22.9652,
 -25.3283, 10.1803, 14.2238, 4.1021, 11.8839, -9.2583, 7.3310, -7.5229,
 7.9154, -26.2988, 2.0203, 2.9490, 4.6372, 10.5871, 14.5873, 8.3466,
 4.3750, -3.4053, 5.8707, -1.8310, -6.1250, 6.1537, 11.2335, 3.7743,
 20.7993, -7.7024, 1.5719, -13.2223, 5.5466, -17.8620, -10.7962, 0.1783,
 -3.9857, -7.8429, 1.3310, 9.5470, -13.4463, 16.9022, 2.0262, -12.7968,
 14.1492, -4.8442, 5.6812, 5.4264, -3.8780, -4.0601, 1.4185, 1.0981,

-----------------------
======================= quant matrix & stored block below
======================= DC scale = 6
 2, 8, 4, 10, 14, 18, 28, 30,
 6, 14, 20, 24, 38, 46, 52, 56,
 4, 16, 34, 48, 58, 70, 84, 96,
 8, 30, 44, 64, 76, 96, 106, 120,
 16, 40, 52, 80, 92, 114, 138, 154,
 16, 46, 70, 92, 118, 142, 160, 190,
 28, 48, 82, 108, 136, 160, 190, 224,
 28, 62, 96, 120, 160, 188, 224, 256,
 57, -45, -15, 18, -26, -22, 21, 7,
 1, 36, -9, 9, 6, 31, -12, 23,
 -25, 10, 14, 4, 12, -9, 7, -8,
 8, -26, 2, 3, 5, 11, 15, 8,
 4, -3, 6, -2, -6, 6, 11, 4,
 21, -8, 2, -13, 6, -18, -11, 0,
 -4, -8, 1, 10, -13, 17, 2, -13,
 14, -5, 6, 5, -4, -4, 1, 1,
======================= qmatrix   2
======================= quant matrix & retrieved block below
======================= DC scale   6
 2, 8, 4, 10, 14, 18, 28, 30,
 6, 14, 20, 24, 38, 46, 52, 56,
 4, 16, 34, 48, 58, 70, 84, 96,
 8, 30, 44, 64, 76, 96, 106, 120,
```

```
 16, 40, 52, 80, 92, 114, 138, 154,
 16, 46, 70, 92, 118, 142, 160, 190,
 28, 48, 82, 108, 136, 160, 190, 224,
 28, 62, 96, 120, 160, 188, 224, 256,
 336, -40, -12, 10, -14, -18, 0, 0,
 0, 28, 0, 0, 0, 0, 0, 0,
 -24, 0, 0, 0, 0, 0, 0, 0,
 8, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0,
 16, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0,
========== qmatrix = 2
----------BLOCK BEFORE & AFTER INVERSE DCT below
 336.0000, -40.0000, -12.0000, 10.0000, -14.0000, -18.0000, 0.0000, 0.0000
 0.0000, 28.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 -24.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 8.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 16.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,

 36.6181, 44.3708, 40.9206, 37.3695, 44.7054, 45.8582, 39.1607, 37.6173,
 32.0919, 40.0006, 36.8388, 33.6644, 41.4081, 42.9377, 36.5287, 35.1415,
 35.6596, 43.8567, 41.2277, 38.7496, 47.2468, 49.4726, 43.5964, 42.4974,
 37.8821, 46.4560, 44.5232, 42.9547, 52.4365, 55.5719, 50.3919, 49.6697,
 32.0712, 41.0529, 39.8737, 39.2897, 49.8372, 53.9572, 49.5307, 49.2164,
 29.7016, 39.0600, 38.5770, 38.9026, 50.4347, 55.4643, 51.7340, 51.7964,
 26.7749, 36.4218, 36.4716, 37.4934, 49.7790, 55.5048, 52.3074, 52.6582,
 17.6565, 27.4594, 27.7975, 29.1962, 41.8896, 47.9922, 45.0831, 45.5899,

----------
   original_bpp   8.00000
   compress_bpp   2.82936
   compress_mse   420.92261
   compress_rms   20.51640
   compress_snr   21.88878
   compress_rat   0.12926
SUCCESSFUL RUN!
```