Summation
May 2009, pp. 1-6
http://ripon.edu/macs/summation

Multivariate Neuroimaging Analysis: An Approach for Multiple Image Modalities

Christopher J. Larsen
Department of Mathematics and Computer Science – Ripon College

***Abstract*—This study presents a multivariate technique for analyzing variable significance in neuroimages. Traditional univariate analysis presents the difficulty of controlling for the familywise error rate, and multivariate analysis across the entire brain of one subject presents computational difficulties. This study proposes a multivariate technique with different imaging modalities as the outcomes. This hybrid approach takes advantage of the possible correlations between imaging modalities. We demonstrate the analysis for a single voxel in the brain and compare image maps from the univariate and multivariate techniques for VBM and fMRI imaging modalities. Our results show that there is evidence that the multivariate technique is a viable method for the researchers toolkit.**

***Key Words*— Alzheimer's disease, brain maps, fMRI, multivariate analysis, neuroimaging, VBM.**

I. Introduction

NEUROIMAGING has become a very important technology in the medical field. It has helped doctors diagnose brain diseases, pinpoint structural damage, and identify disease related changes. Neuroimaging has also enhanced the abilities of medical researchers. Medical researchers are now able to examine the brain more closely in hopes of detecting brain diseases earlier. By conducting experiments and analyzing results, researchers hope to identify differences in brain structure and function associated with diseases earlier in peoples' lives. If they are able to do this, it may be possible to treat these diseases earlier with the ultimate goal of prevention.

The techniques used to analyze the brain images play an important part in the researchers' ability to identify significant results. Statistical parametric mapping (SPM) is used to identify functionally specialized brain responses [1]. The results of the SPM can then be analyzed further to test for significant results of certain explanatory variables. We propose the use of multivariate techniques and demonstrate its use within the context of early detection of Alzheimer's disease, specifically the impact of family history on brain function and structure.

We proceed by briefly reviewing some standard approaches to imaging analysis in section II. In section III we lay out how to employ a mass-multivariate approach to image analysis. We demonstrate the procedure for on a real data set in section IV, and discuss the findings and possible future directions in section V.

II. Current Approach To Imaging Analysis

Raw imaging data goes through several different procedures before it is ready for the implementation of advanced imaging analysis. The data must first be realigned to get the primary axes of each image as close to the same as possible. The data are then normalized to account for different skull and brain shapes and sizes, and each image is sized to one representative brain. The data are then spatially smoothed using a Gaussian kernel. A design matrix is then applied to a general linear model to produce level 1 data, which shows if the change in the voxel was significant when a single subject was on and off the task. It is from this level 1 data that more advanced imaging analysis can be conducted, like in identifying regions of the brain that family history of Alzheimer's disease has a significant effect.

The most common method for analyzing imaging data is the mass-univariate approach. Friston [1] writes, "Over the years statistical parametric mapping has come to refer to the conjoint use of the general linear model (GLM) and Gaussian random field (GRF) theory to analyze and make classical inferences about spatially extended data through statistical parametric maps (SPMs)." This approach uses one imaging modality (e.g. fMRI) and applies a linear model at every point in the brain to obtain level 1 data and another linear model at every point in the brain to assess level 2 hypotheses across subjects.

The drawback in applying mass-univariate models to the data is that the familywise error rate must be controlled. The univariate model is being applied to hundreds of thousands of correlated test statistics. This presents a multiple comparison issue. The familywise error rate, the chance of any false positives, is the standard measure of Type I errors in multiple testing [2]. In order to properly control for Type I errors in the univariate analysis, test statistics would have to be so significant that it is likely no significant results would be found.

There are several methods for controlling for familywise

Work presented to the college May 12, 2008. This work was supported in part by the Ripon College Mathematics and Computer Science Department and Dr. Sterling Johnson, head of Geriatrics, University of Wisconsin Madison Medical School.

C. J. Larsen hopes to work as a Chief Financial Officer or begin his own business, Ripon, WI 54971 USA (phone: 763-286-9165; e-mail: larsen.christopherj@ gmail.com).

error including Bonferroni methods, permutation methods and random field theory. Improvements in these methods and in computational power have made controlling for multiple comparison issues more realistic. However, the Bonferroni-like methods become so conservative that it is almost impossible to detect any affects. The permutation method shows improvements over random field theory for low smoothness and low degrees of freedom. Random field theory would be the best method for controlling for familywise error in the extremely limiting case where the observations have a perfect normal multivariate distribution. However, this case is highly extremely limiting, suggesting alternative methods are appropriate. These observations are noted by Nichols and Hayasaka [2], 2003. They also note a suspected conservativeness of random field theory methods.

A strategy to eliminate threshholding requirements is to apply multivariate analysis to the data. The multivariate technique would allow researchers to identify if a certain parameter was significant for an entire brain. However, there are computational complications that arise. At the very latest, these computational complications would arise when attempting to invert the full model sum of squared errors matrix that is hundreds of thousands of rows by hundreds of thousands columns. Another issue that accompanies multivariate analysis is its inability to support inferences about regionally specific effects [1].

However, it is possible to take the multivariate analysis in a different direction. Instead of using multivariate analysis to test the whole brain in one pass for a single modality, one could apply a mass-multivariate analysis incorporating the differing modalities (e.g. fMRI and VBM). The two or more imaging modalities would be used as a matrix of outcome variables. The test could then be run independently for each voxel in the brain. The outcome matrix would have dimensions of the number of subjects by the number of imaging modalities being compared. This would produce a sum of squared errors matrix that would be square with dimensions equal to the number of imaging modalities in the multivariate analysis. This matrix would be easily invertible, and the computational complication would be eliminated.

Intuitively, it makes sense to compare different imaging modalities in a multivariate fashion. Take for example a test on the significance of family history of Alzheimer's disease as it pertains to brain function. Testing this univariately using the fMRI modality may not produce significance results, and testing it univariately using the VBM modality may not produce significant results. However, accounting for the correlation between the two modalities using multivariate analysis may produce significant results. Intuitively, if there is no grey matter in an area of the brain, there will be no function in that area. Thus, there is likely a correlation between VBM and fMRI modalities for which one should account. Empirical evidence for such a correlation is presented by Oakes [3], 2007.

We hypothesize that accounting for correlations between imaging modalities using multivariate analysis may produce more power than univariate tests alone when testing for variable significance on imaging outcomes. We will test our hypothesis empirically by testing for the significance of family history of Alzheimer's disease in the univariate case and multivariate case for fMRI and VBM images.

III. Multivariate Approach to Imaging Analysis

The theoretical approach to carrying out a multivariate test is quite similar to the univariate technique. The approach begins with the full model containing all of the explanatory variables. The full model explanatory variables are contained in the design matrix, $\mathbf{X}_f$. The outcome variable matrix, $\mathbf{Y}$, contains columns of outcomes from different imaging modalities in this study. The full model is

$$\mathbf{Y} = \mathbf{X}_f \boldsymbol{\beta}. \tag{1}$$

The coefficient matrix, $\boldsymbol{\beta}$, is estimated using the same formula as in the univariate case, i.e.

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}_f{}^T \mathbf{X}_f)^{-1} \mathbf{X}_f{}^T \mathbf{Y}. \tag{2}$$

Each column of the design matrix in the multivariate case will be exactly equivalent to the univariate design matrix. The predicted values, $\hat{\mathbf{Y}}$, is simply attained from the design matrix and the estimated coefficient matrix via $\hat{\mathbf{Y}} = \mathbf{X}\hat{\boldsymbol{\beta}}$ for both the full and reduced models.

After the predicted values for both the full and reduced models are obtained, residual matrices, $\mathbf{ER}_f$ for the full model and $\mathbf{ER}_r$ for the reduced model, are calculated. A residual matrix is calculated by subtracting the predicted outcome matrix, $\hat{\mathbf{Y}}$, from the actual outcome matrix, $\mathbf{Y}$. The residual matrix is then used to calculate a sum of squared errors matrix. A sum of squared errors matrix is calculated by multiplying the transpose of a residual matrix by the residual matrix, denoted $\mathbf{SSE}_f$ for the full model sum of squared error matrix and $\mathbf{SSE}_r$ for the reduced model sum of squared error matrix,

$$\mathbf{SSE}_f = \mathbf{ER}_f{}^T \times \mathbf{ER}_f, \tag{3}$$

$$\mathbf{SSE}_r = \mathbf{ER}_r{}^T \times \mathbf{ER}_r. \tag{4}$$

The sum of squared error matrix in the multivariate analysis will contain the sum of squared errors for the corresponding univariate analysis along the main diagonal and the cross products of the residuals of the different outcome variables in the off diagonal. Taking into account the possible correlation between the multivariate outcomes in the error matrix may lead to more power than the univariate case.

After obtaining the sum of squared errors matrix for both full and reduced model, a multivariate analysis of variance test is carried out to test the significance of removed variables.

This is completed by calculating an F-statistic. The F-statistic calculation begins in the same manner as in the univariate case. Let

$$\mathbf{H} = \mathbf{SSE}_r - \mathbf{SSE}_f, \tag{5}$$

and

$$\mathbf{E} = \mathbf{SSE}_f. \tag{6}$$

Equation (5) is scaled by (6) by taking the inverse of (6) and multiplying it by (5). Then

$$\mathbf{T} = \mathbf{E}^{-1}\mathbf{H}.$$

Once **T** is obtained, there are several different methods that can be used to calculate a multivariate F-statistic. This study uses Pillai's Trace method for calculating the F-statistic.

Let $\lambda_1, \lambda_2, \ldots, \lambda_k$, be the k eigenvalues of **T**. Then let

$$v = \sum_{i=1}^{k} \lambda_i / (1 + \lambda_i), \tag{7}$$

and

p = rank(SSE_r)
q = number of reduced model parameters
s = min(p,q)
N = number of observations
w = N minus number of full model parameters
m = 0.5(|p-q| - 1)
n = 0.5(w – p – 1).

Then, the statistic

$$F = \frac{2n + s + 1}{2m + s + 1} * \frac{v}{s - v} \tag{8}$$

follows Fisher's F-distribution with s(2m + s + 1) numerator degrees of freedom, and s(2n + s + 1) denominator degrees of freedom under the assumption of Gaussian errors.

If the p-value of the F-statistic is less than our significance level, the explanatory variables removed from the model are significant in the multivariate analysis.

IV. Results

Level 1 data from 23 subjects was used to test the significance of family history of Alzheimer's disease. The outcome data was from fMRI and VBM imaging modalities. Age and education years were also controlled for in our model. Family history of Alzheimer's disease was removed in the reduced models. Testing for the significance of family history was conducted using both the univariate cases and using the multivariate analysis.

The two full models in the univariate analyses are

$$fMRI = \beta_0 + \beta_1 age + \beta_2 edyrs + \beta_3 FHx1 + \beta_4 FHx2 + \beta_5 FHx3 + e, \tag{9}$$

and

$$VBM = \beta_0 + \beta_1 age + \beta_2 edyrs + \beta_3 FHx1 + \beta_4 FHx2 + \beta_5 FHx3 + e. \tag{10}$$

The two reduced models in the univariate analyses are

$$fMRI = \beta_0 + \beta_1 age + \beta_2 educyrs + e, \tag{11}$$

and

$$VBM = \beta_0 + \beta_1 age + \beta_2 educyrs + e. \tag{12}$$

Testing for significance of family history is straight forward in the univariate cases. The sum of squared errors are found for the reduced and full models, an F-statistic is calculated, and the p-value is found. The F-statistic is calculated by

$$F = \frac{SSE_R - SSE_F}{SSE_F} \times \frac{N - p}{p - q}, \tag{13}$$

where N is the number of observations, p is the number of full model parameters, q is the number of reduced model parameters, and SSE_R and SSE_F are the resultant sum of squared errors for the reduced and full models respectively.

The F-statistics and p-values for the fMRI and VBM models testing for the significance of family history are

$$F_{fMRI} = \frac{21.65 - 15.52}{15.52} * \frac{23 - 6}{6 - 3} = 2.237$$
$$p - value_{fMRI} = 0.121$$

$$F_{VBM} = \frac{23.78 - 17.61}{17.61} * \frac{23 - 6}{6 - 3} = 1.99$$
$$p - value_{VBM} = 0.154$$

In the multivariate analysis, there is only one full and one reduced model. The single outcome variables in the univariate analyses are combined in to one outcome matrix. The full multivariate model is

$$[fMRI \quad VBM] = [1 \quad age \quad edyrs \quad FHx1 \quad FHx2 \quad FHx3][\boldsymbol{\beta}]. \tag{14}$$

The reduced multivariate model is

$$[fMRI \quad VBM] = [1 \quad age \quad edyrs][\boldsymbol{\beta}]. \tag{15}$$

It is important to note the sum of squared errors matrices of the full and reduced models. The sum of squared errors matrices are

$$\mathbf{SSE}_f = \begin{bmatrix} 15.52 & 9.23 \\ 9.23 & 17.61 \end{bmatrix}, \quad (16)$$

$$\mathbf{SSE}_r = \begin{bmatrix} 21.64 & 4.93 \\ 4.93 & 23.78 \end{bmatrix}, \quad (17)$$

The main diagonals on these matrices are equivalent to their respective univariate sum of squared errors. The off-diagonals are the cross-products of the residuals, which indicate if there is correlation between the residuals of the two outcomes.

Having found the sum of squared errors matrices for both the full and reduced multivariate models, we can easily calculate the **H** and **E** matrices. **H** is (17) minus (16), using matrix subtraction, and **E** is simply (16). The **H** and **E** are used to obtain matrix **T**, which is the inverse of **E** multiplied by **H**. In our example we have

$$\mathbf{T} = \begin{bmatrix} 0.736 & 0.703 \\ -0.677 & 0.711 \end{bmatrix}, \quad (18)$$

Now that **T** has been calculated, Pillai's Trace will be used to calculate the F-statistic and p-value for the multivariate analysis. The first step is obtaining the two eigenvalues of **T** which turn out to be $\lambda_1 = 1.432$ and $\lambda_2 = 0.072$. The eigenvalues are entered into (7) as well as the values for the other elements of Pillai's Trace, is

$v = 0.656$ $N = 23$
$p = 2$ $w = 17$
$q = 3$ $m = 0$
$s = 2$ $n = 7.$

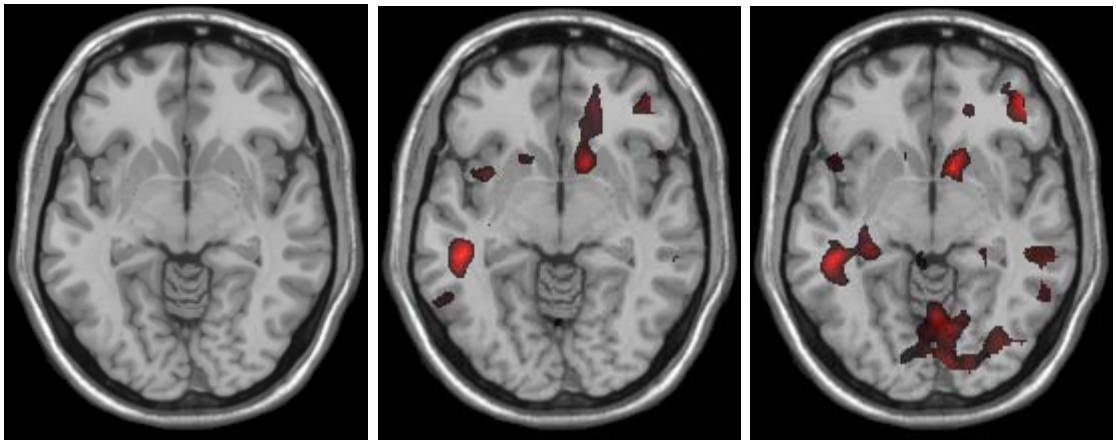
The F-statistic and p-value, with 6 numerator degrees of freedom and 34 denominator degrees of freedom, for the multivariate can be obtained by entering the appropriate values into (8).

$$F_{MV} = \frac{2*7+2+1}{2*0+2+1} * \frac{0.656}{2-0.656} = 2.77$$
$$p - value_{MV} = 0.027.$$

We can see that when testing the statistical significance of family history in the two univariate cases, as it pertains to brain function and structure, neither of the univariate cases produce statistically significant results. However, using the multivariate analysis to account for the possible interaction between brain function and structure, family history is statistically significant at the 5 percent level. Intuitively, this makes sense. If there is no grey matter to detect using the VBM modality in certain areas of the brain, one certainly will not detect functionality in those areas. Thus, the multivariate analysis seems more appropriate. This result would suggest that the univariate method of analysis may sometimes miss the statistical significance of explanatory variables.

The previous example was for only one voxel of the brain. The test is repeated for every voxel in the brain in order to obtain a map of all the points in the brain showing significant results of family history using the multivariate analysis. We wrote code to perform this analysis for every point in the brain using the program *R*. The code can be found in the appendix of this paper.

The brain map that is produced using the multivariate analysis can be compared to the maps of the univariate analyses of the fMRI and VBM modalities. Figure 1 is an image of the Level 2 analyses on the significance of family history of Alzheimer's disease and brain function and structure. The image on the left is the univariate analysis with VBM as the outcome. The center image is the univariate analysis with fMRI as the outcome. The figure on the right represents the multivariate analysis.



A B C

Figure 1. Fire maps of the univariate analyses and multivariate analysis. From left to right the maps represent: univariate analysis with VBM as outcome (A), univariate analysis with fMRI as outcome (B), and multivariate analysis (C).

The maps in Fig. 1 represent the same slice of the brain. We can see from the maps that there is a significant difference in the amount of firing between the images. The univariate VBM map shows almost no statistically significant areas. The univariate fMRI map shows two areas where there is a significant firing. The multivariate map shows significant results in the same two areas as the fMRI map, and an additional third area that shows significant firing. There is also two areas in the back of the brain that show areas of firing, but these areas do not show highly significant results.

V. Summary

This study has shown that there is basis to consider using a multivariate analysis between imaging modalities when analyzing the significance of variables on brain function and structure. Intuitively, this type of analysis makes sense because there will likely be no function if there is no grey matter in a particular area. Analytically, there are difficulties controlling for familywise error in the univariate analysis. Multivariate analysis across the entire brain results in computational difficulties. However, when using multivariate analysis across imaging modalities, the computational difficulties of multivariate analysis across the entire brain are eliminated.

Empirically, this study has shown that it is possible for the univariate analysis with two different outcome modalities to produce insignificant results, but the multivariate analysis between the two modalities at the same voxel to produce significant results. When comparing the three different imaging maps, there is also a noticeable difference in the firing patterns.

Although we have shown that using the multivariate analysis can produce substantially different results than using the univariate analysis alone, further investigation into power of this type of multivariate analysis to detect results differing from the univariate analysis is warranted. Also, this type of analysis does not tell us how the different modalities are correlated. Nonetheless, the multivariate analysis provides important advantages over the univariate analysis.

Acknowledgment

First I would like to thank my colleagues, Nicholas Krueger and Brett Wegner. It is only through the combination of our efforts that this work is possible. I would also like to thank Timothy Hess for the countless hours he has contributed in the development and implementation of this study. Our collaboration with Dr. Hess has made the results of this study valuable both for those inside and outside the area of mathematics.

I would also like to extend acknowledgment to the professors who have imparted their mathematical knowledge on me, David Scott, Karl Beres, and Norm Loomer. Their dedication and passion to educating students is rivaled by few. And to the rest of the Mathematics and Computer Science Department, Diane Beres and Kristine Peters, their involvement in the Senior Seminar experience has been integral to learning valuable skills that can be utilized in many different facets of life.

References

[1] K. J. Friston, "Introduction: Experimental Design and Statistical Parametric Mapping," *Human Brain Function*, Academic Press, ed. 2, 2003.

[2] T. Nichols and S. Hayasaka, "Controlling the familywise error rate in functional neuroimaging: a comparative review," *Statistical Methods in Medical Research*, vol. 12, pp. 419-446, 2003.

[3] T. R. Oakes, A. S. Fox, T. Johnstone, M. K. Chung, N. Kalin, R. J. Davidson, "Integrating VBM into the General Linear Model with voxelwise anatomical covariates," *Neuroimage*, vol. 34, ed. 2, pp. 500-508, Jan. 15, 2007.

Christopher J. Larsen was born in St. Cloud, MN, December 24, 1986. Larsen began his college career at the age of 16 at the Cambridge campus of Anoka-Ramsey Community College. Larsen then continued his college career at Ripon College where he was awarded a full-tuition Pickard Scholarship. Larsen graduated on from Ripon College, Ripon, WI, U.S., on May 16, 2009, obtaining his Artium Baccalaureus (Bachelor of Arts) with majors in business administration, mathematics, and economics, and a minor in French..

During his undergraduate career at Ripon College, he worked as an Economics Departmental Assistant. He was an Intern at Diverse Option, Inc., a local nonprofit organization, where he helped develop an Emergency Action Plan and developed, collected, analyzed, and presented the results of several stakeholder surveys. The surveys were used to guide better prepared the organization for the future. He also was the Director of the Creative Enterprise Center, a student operated consulting business dedicated to providing services to small business owners and entrepreneurs. He will be attending the Simon School of Business at the University of Rochester, Rochester, NY, in the fall of 2009 to obtain his M.B.A.

Mr. Larsen is a member of Phi Beta Kappa, Laurel Society, Omicron Delta Epsilon, and Phi Sigma Iota National Honor Societies, the Sigma Chi Fraternity, and Students in Free Enterprise. Mr. Larsen enjoys restoring classic cars, sports, bowling, and outdoor activities.

Appendix

The *R* code used to run the multivariate analysis appears below.

```
multi.image<-
function(formF,formR,img.dir,img.ext,data)
{

  img.files<-
list.files(img.dir[1],img.ext,full.names=T
)
  tmp.array<-
f.read.nifti.volume(img.files[1])
  out.dim<-dim(tmp.array)
  out.array<-array(0,out.dim)
  mmf<-
as.matrix(model.frame(formF,data=data))
  mmr<-
as.matrix(model.frame(formR,data=data))
  rm(tmp.array)
  for(k in 1:out.dim[3]){#  for(k in 1){
    for (u in 1:length(img.dir)){
```

```
      img.files<-
list.files(img.dir[u],img.ext,full.names=T
)
      if(u==1) image.array<-
array(0,c(out.dim[1:2],length(img.files),l
ength(img.dir)))
      for(n.sub in 1:length(img.files)){
        print(n.sub)
        image.array[,,n.sub,u]<-
f.read.nifti.slice(img.files[n.sub],k,1)[1
:out.dim[1],1:out.dim[2]]
      }
    }
    print(k)
    for(i in 1:out.dim[1]){
      for(j in 1:out.dim[2]){
        Y<-image.array[i,j,,]
        if( ! any(is.na(Y))){
           FF<-lm(Y~mmf)
           FR<-lm(Y~mmr)
           out.array[i,j,k]<-
anova(FF,FR)[2,"Pr(>F)"]

        }
      }
    }
  }
  out<<-out.array
}
```

The data used to conduct the multivariate analysis in this paper appears in Table 1.

TABLE I
EMPIRICAL DATA

FamHx	Edyrs	AgeAtV1	fMRI	VBM
2	12	900	-0.355	-0.161
1	16	875	-0.600	0.133
0	18	919	-0.588	-0.294
2	12	925	0.280	0.120
0	12	1,024	-0.267	-0.067
2	18	799	-0.222	-0.111
0	18	775	0.043	0.043
0	12	912	0.211	0.211
1	12	776	-1.000	0.000
2	18	811	-0.308	-0.038
1	12	933	-0.107	-0.071
2	20	749	0.000	0.059
0	17	817	0.250	-0.111
0	14	929	0.214	0.000
3	18	960	-0.385	-0.115
0	20	939	0.000	-0.081
0	18	978	-0.154	-0.077
0	12	792	0.286	-0.214
0	16	919	0.217	0.043
3	21	880	0.538	-0.154
3	14	868	0.154	-0.077
3	16	761	0.194	0.056
3	16	800	0.588	0.059
3	18	846	0.222	0.037
3	20	908	-0.148	0.074
3	16	904	-0.269	0.077
3	13	772	-0.107	-0.143
3	18	777	0.333	0.037
3	20	829	-0.308	-0.038
3	16	974	-0.296	-0.037

FamHx = family history of Alzheimer's disease, Edyrs = years of education, AgeAtV1 = age in months at time of scan, fMRI = fMRI covariates, VBM = VBM covariates.

Summation
May 2009, pp. 7-14
http://ripon.edu/macs/summation

The Dominance Analysis Approach for Comparing Predictors in Neuroimaging

Nicholas R. Krueger and Brett M. Wegner
Department of Mathematics and Computer Science – Ripon College

***Abstract*—The Dominance Analysis approach for ranking the relative importance of predictors in multiple regression has never before been applied to the field of neuroimaging. This method is useful to researchers who hope to identify variables likely to significantly influence brain function. It uses the same data already used to generate Statistical Parametric Maps and provides a useful and intuitive interpretation of relative importance. The process could potentially save researchers time and money in neuroimaging studies by helping them focus their studies on the likeliest predictor variables.**

***Key Words*—Complete dominance, Conditional dominance, General dominance, Multiple regression, Neuroimaging.**

I. Introduction

According to the latest statistics from the Alzheimer's Association, a nation-wide support and research organization, as many as 5.3 million people are living today in the United States as Alzheimer's Disease (AD) patients. The combined direct and indirect costs of AD and other dementias to Medicare, Medicaid, and other businesses total approximately $148 billion per year. Every 70 seconds, someone develops AD – bad news, given that this affliction is the sixth-leading cause of death in this nation [1]. There are few treatments available and no cures, and until recently, the outlook has appeared bleak for any significant progress in slowing or reversing these ugly trends.

However, with the advent of the most modern medical and research technologies has come new hope. The University of Wisconsin-Madison's Institute on Aging, which opened its doors in 1973, is dedicated to addressing health concerns associated with later life, including diseases and impairments like AD. Through the use of the latest methods in neuroimaging and the measurement of brain structure and function, researchers there are working to learn more about the effects of AD and such other similarly debilitating ailments as dementia and traumatic brain injury. Dr. Sterling Johnson, Associate Professor in the Department of Medicine at the University, specifically explains his mission:

> "In my lab we use brain imaging in conjunction with neuropsychological measurement to study cognitive disorders of memory and self-awareness. The lab seeks to address questions such as: How early in life does Alzheimer Disease begin and can functional imaging provide new knowledge about presymptomatic disease progression? What are the neural substrates of awareness (or unawareness) of deficit in AD and other cognitive disorders? To answer questions like these we collaborate with a talented multi-disciplinary team, and study healthy volunteers across the age spectrum with and without risk factors for dementia, patients with Alzheimer's disease, Mild Cognitive Impairment, frontal temporal dementias, and patients with traumatic brain injury [2]."

This work – being conducted across the globe at numerous research institutions despite being agonizingly slow, labor-intensive, and extraordinarily expensive – may one day produce new knowledge that could lead to breakthroughs in treatment and prevention of diseases like AD. If researchers can begin to understand which parts of the brain are responsible for certain activities, and if they can ultimately determine any differences between healthy brains and those of AD patients, it is possible that they will be able to find early warning signs of such diseases and draw conclusions that could save thousands of lives.

II. Project Objectives

The sheer volume of data recorded in the course of neuroimaging research is overwhelming. Though the medical researchers at UW-Madison and elsewhere are experts in the science of gathering data and designing studies to measure potentially significant differences in brain function, they realize that their efforts may be hindered if they are unable to adequately analyze the advanced information they have collected. Researchers are willing to spend years of time and hundreds of thousands of dollars designing experiments and collecting accurate data; they realize that similarly detailed attention to the data analysis portion of the project is equally critical. (To put this in perspective, consider that Yale University recently installed a new 3.0 Tesla MRI scanner at a cost of approximately $1.8 million [3]; the going rate for the

Work presented publicly to the college 24-26 March 2009.

N.R. Krueger will enroll at Georgetown University in August 2009 to pursue a Master of Arts degree in International Relations following graduation from Ripon College, Ripon, WI. (E-mail: kruegern@ripon.edu).

B.M. Wegner will enroll at the University of Wisconsin-Madison following graduation from Ripon College to pursue a Masters degree in Accounting (E-mail: wegnerb@ripon.edu).

cost of a single one-hour scan session seems to hover between $500-1000.)

During a visit by the authors to the research laboratories at the University of Wisconsin, Dr. Johnson explained a few obstacles faced daily by his team for which he would welcome some assistance. These require further explanation later, but it will suffice here to state two simple yet grand objectives of our undergraduate mathematics team:

1) Given the difficulty and expense of gathering brain functioning data through neuroimaging, we sought to determine and apply an appropriate statistical method by which research variables can be ranked according to relative significance. Such a ranking is useful primarily as a tool for neuroimaging researchers to focus and guide their research priorities.
2) Once we determined an appropriate method, we wanted to build a computer interface that could automatically apply it and that was relatively straightforward for clinical investigators to use. Such a program, once implemented, would be a major timesaver for researchers hoping to narrow their efforts.

III. The Data Collection Process: Understanding the Objectives

It is not the primary purpose of this project to provide a full and detailed explanation of the data gathering process carried out by research teams. It is not our purpose either to evaluate the utility of certain data preprocessing techniques; that is beyond the scope of this project, though admittedly, much work remains to be done in the theory behind these adjustments and in analysis of their effectiveness. However, to fully understand our proposed method and the context of this analysis, several stages of the data collection process must be briefly described.

In a typical study, researchers will have generated a hypothesis to be tested. Perhaps they believe that a certain region of a healthy brain is likely to have a different proportion of grey matter to white matter than the same region in the brain of an AD patient; perhaps they believe that the difference between AD and non-AD brains will not be a matter of structure but one of volume of blood flowing to a certain region during the exercise of that region; the possibilities are limitless. Depending on this hypothesis, the researchers choose an imaging modality equipped to make the measurement to be considered.

Some of the most common modalities used are Magnetic Resonance Imaging (MRI) scans and functional Magnetic Resonance Imaging (fMRI) scans. A standard MRI scan is used to build a high-resolution, highly detailed image of tissue structure. FMRI scans, instead of measuring tissue structure, basically measure blood flow by reading the magnetic output of oxygenated blood contrasted with deoxygenated blood. The presumption is that oxygen-rich blood flows freely to parts of the brain being used at any given time; by measuring the amount of blood present at any given location, it is thought that one is really gauging functional activity in that region [4].

During the course of a fMRI scan, the machine takes readings of blood flow in the brain in three dimensions, creating cubic regions called *voxels* (just as a digital camera generates two-dimensional pixels, this machine generates three-dimensional volumetric pixels), each of which measures approximately 2 mm^3 in volume. Thus, a fMRI scan records a measurement of blood flow at a certain spatial region (each voxel) at a certain time. In the course of 20-30 seconds, this measurement is recorded at every (voxel) within the brain; this process is repeated for the duration of the scan, which might last 15-30 minutes, depending on the experiment [4].

The authors observed this process occurring as the research team calibrated its fMRI scanner in our presence. We quickly realized, as we watched data streaming into the team's computers, that the volume of data being recorded was enormous. The fMRI scanner was taking approximately 150 measurements in each of three dimensions, generating hundreds of thousands of tiny voxels within the brain, each of which was being measured at least 60 times during a scan. For any one scan subject, then, there is clearly no shortage of raw reference points from which to draw inferences about what is going on within that one person's brain.

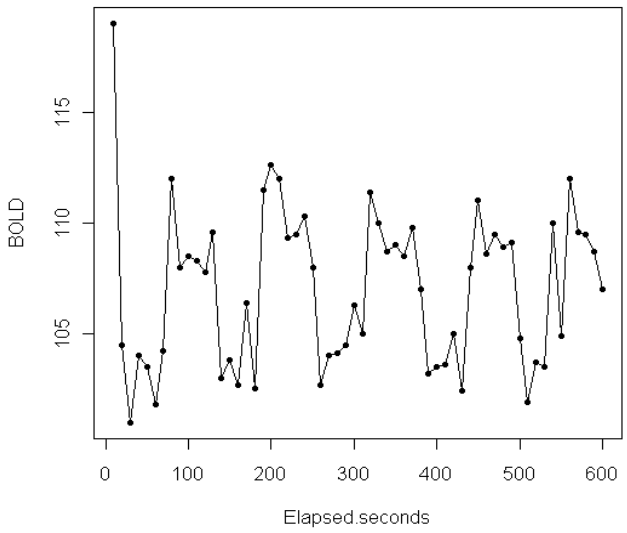


Figure 1. A sample measurement of one voxel over time. Researchers will assess whether these data match a given stimulus or is merely random.

Figure 1 is a possible example of a very simple fMRI experiment. The graph illustrates a measurement of blood flow at one voxel over a period of ten minutes. In this particular case, the test subject was asked to listen and focus on words being read aloud for some seconds before being allowed to rest; this was repeated five times. The researchers, then, would ultimately perform statistical analysis on the data from this graph to determine whether the measured peaks and valleys significantly correspond with the provided stimuli.

At this point, a very large table of data has been recorded. But there are many things the researchers do to this raw data

set to prepare it for analysis. For example, one major source of measurement error is the potential for a test subject to physically move his or her head during the scan procedure. This is obviously a major problem, since the measurements being taken are precise to within a few millimeters. The involuntary movement caused by a test subject's breathing or even his arterial movements and pulsations can be problematic. Any such movement errors add up in the form of additional residual variance for each measurement, all of which potentially reduce the accuracy and usefulness of the linear model to be generated later.

How much motion is significant? The answer may be surprising: it depends on what moves. Deep inside the brain, signals are relatively uniform and vary minimally from voxel to voxel; ultimately, a millimeter of movement causes an amount of signal change similar to the signal change caused by the magnetism of the flowing blood. Said differently, movement of one millimeter has a false effect on the fMRI measurement very similar to a genuine effect. At the edge of the brain, however, where signals are not as uniform, the problem is magnified so badly that movement of only a few microns causes a similarly false effect [5]. This effect has to be accounted for with movement parameters in any linear model. Fortunately, because conclusions are ultimately drawn only from detected signals that are correlated with task performance within the scanner, the vast majority of these false positives are ignored by the model as random error; however, such error has the effect of making it more difficult to detect the genuine effects [5].

Motion correction is only one of several similar, physical adjustments to the raw data. Another important step is the normalization of motion-corrected data to a standard size and shape of brain. That is, different people have differently sized brains; in order for any kind of consistent comparison to take place among numerous test subjects, it is necessary to account for this. The idea is to match *functionally* equivalent sections of brain among multiple test subjects and to ignore purely anatomical and topological differences; this step is helpful in permitting researchers to extrapolate findings from among only a few test subjects to a population at large [5].

After the data has been satisfactorily adjusted to allow for whatever effects (motion, spatial normalization, and potentially others) the researchers might be concerned about, it is finally ready for statistical analysis. The approach we take is to build a design matrix X out of n possible predictor variables in order to create a general linear model of the form

$$Y(s) = X \cdot \beta(s) + \varepsilon(s)$$

where $Y(s)$ is the n-vector of observed values recorded at voxel s; X is a fixed design matrix of predetermined independent variables; $\beta(s)$ is the vector of coefficients that modify X; and $\varepsilon(s)$ is a vector of observed error values. Using some basic principles of linear algebra, we calculate an estimate of $\beta(s)$ that minimizes error in the model via ordinary least squares regression; this process guarantees that the specific linear model generated is the one that minimizes overall error. That is to say that we build a model such that any voxel $Y(s)$ can be represented as a linear combination of predictor variables and coefficients:

$$Y(s) = 1 \cdot \beta_0(s) + X_1\beta_1(s) + X_2\beta_2(s) + \ldots + X_n\beta_n(s) + \varepsilon(s)$$

in which X_i represents the vector of a predictor variable, **1** is a column vector of 1's, and our estimate of the coefficients that minimizes the overall sum of squared error is given by

$$\hat{\beta}(s) = (X'X)^{-1} X'Y(s)$$

Once this estimate of $\beta(s)$ is known, it is possible to test the effect of any given predictor variable by comparing two similar linear models and generating a test statistic by which it can be determined whether or not the data differs significantly according to that given predictor. In the examples considered above, it might be possible to conclude that concentrating and listening to the words read aloud cause specific regions of the brain to "light up" with activity, thus lending insight into how the brain functions to perform that given task.

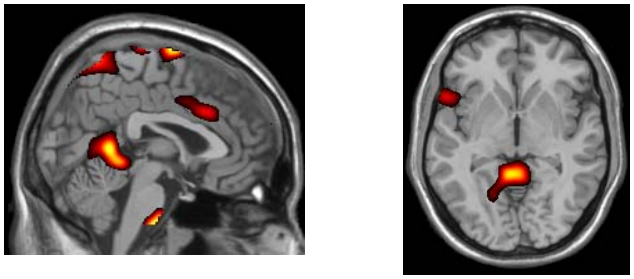


Figure 2. Statistical Parametric Maps from Sample Data.

This modeling, compiled from all the data collected across one brain, can be collected to produce three-dimensional pictures that illustrate what is happening within one brain in a process known as Statistical Parametric Mapping. In the images shown, the colored regions represent areas of the brain that were concluded in such an analysis to be responsible for the thinking functions associated with some stimulus.

To this point, all the analysis described has been applied to the data from only one test subject in what we will refer to as Level 1 analysis. But in order to apply any Level 1 conclusions to the human population at large, similar tests and models must be applied to data obtained from multiple test subjects. If this Level 2 analysis indicates statistically significant differences between, for example, healthy persons and AD patients, researchers may well find the early warning signs of disease that they are looking for.

IV. The Problem

So far, it has been assumed that the linear model produced to explain the data was built from a known list of variables deemed important by the researchers. But intuitively, we recognize that any linear model is only as good as the variables assigned to it. Conclusions can be mathematically accurate and statistically valid but based on erroneous

assumptions or incomplete information, which naturally compromises the validity of such otherwise-correct inferences. In fact, the field of neuroimaging and the mapping of brain activity are quite young. Frequently, the researchers themselves can only make an educated guess as to which predictor variables to include in the design matrices used to generate the linear models, from which any and all conclusions are derived. Though perhaps the main purpose of generating linear models in many contexts is simply to establish the "best," most predictive, and most complete model of a given event, it turns out that the same linear models can be used to compare the usefulness of the predictors themselves.

This is fortunate, because in the neuroimaging context, it is at least as important to the researchers to discover which variables contribute most to their models as it is to determine whether or not their models are 100% complete in accounting for all of their observations. Any conclusions that researchers can use to narrow their efforts and to suggest which effects or which predictors might be more valuable to study than others could result in real savings in time and money. This is especially true given the great expense of obtaining data from enough test subjects to conduct sufficiently powerful Level 2 analysis.

In 2003, Azen suggested a general method for comparing relative importance of predictors in multiple regression that introduces several levels of intuitive and meaningful measures by which such consistency can be established [6]. This procedure, a modification of an earlier procedure called dominance analysis, may be useful for providing neuroimaging researchers a way of estimating the relative importance of all predictor variables included in a linear model.

V. Correlation Measures To Be Used

$R^2_{Y\cdot Xi.}$ In our analysis, we refer to various measures of variable correlation. Azen [6] suggests that one of the most useful measures of importance is the squared product-moment correlation, $R^2_{Y\cdot Xi}$, between the response variable and the explanatory or predictor variables. This value may also be referred to as the squared standardized regression coefficient, found using the regression of Y on X_i alone. The value represents the total response variation that can be explained by the explanatory variable(s) in the model. R-squared values fall in a range between zero and one. This is due to the fact that they are a proportion of the total variance in the model. There is a portion of the total response variation that is explained by the variable(s) and a portion that is not. If all the variation is explained, we will find an r-squared value of one. If none of it is explained by the predictor(s), we will have a value of zero.

$$R^2 = \frac{\hat{\beta}' X' Y - \frac{(Y'1)^2}{n}}{Y'Y - \frac{(Y'1)^2}{n}} \qquad (1)$$

Equation 1 shows how to mathematically calculate an r-squared value. It is simply the ratio of explained error divided by the total model error. This will give us a number that falls in our allotted range. When multiplied by 100, we discover the r-squared values turn into percentages of the total variation.

When all predictors are uncorrelated, we find that the sum of the squared correlations is exactly equal to the total variation explained by all predictors in our model. However, it is rarely the case that one finds predictor variables that are uncorrelated. After factoring in this correlation between the predictor variables, we find that the relative importance of the predictors tends to change depending upon exactly which variables are being compared. Often, the difference in importance is dramatic. For example, consider a situation involving Y, X_1, X_2, and X_3 in which we want to model Y using these three predictor "X" values. Also, suppose that X_2 and X_3 are highly correlated with each other. So too, Y is highly correlated with both X_2 and X_3. However, neither the outcome variable nor the other two predictor variables are highly correlated with X_1. If our regression model included X_2, then there would be little additional contribution made to the prediction of Y if we were to add X_3. In this case, X_3 would seem to be "less important" or have less of a contribution to the model than the uncorrelated X_1. This would work out the same way if we added X_3 to our model first and then decided to add X_2. Researchers [7] have shown that regressions involving two or more correlated predictors may have squared correlations between the observed and predicted values, $R^2_{Y\cdot Y'}$, larger than the sum of the squared correlations between the respective correlated predictors and the observed outcome. This contradictory situation forces us to take into account the effects that such intercorrelations may have on our order of relative importance among the predictor variables.

VI. Dominance Analysis Explained

Budescu's approach [8], called dominance analysis (DA), considers all possible models and subset models for which the comparison of each pair of predictors is relevant. This approach says that a variable is a more important predictor than another if it would be chosen over the other in all possible subset models that could contain either of the predictor variables. This is one of the concepts of the approach that is often misunderstood. You can only quantify the importance level based upon which set of predictor variables you have chosen to include in your analysis. Dominance analysis provides merely a general context due to the fact that it includes all theoretically possible subset models, as well as any that may be of specific interest to the researcher or might be formed using the combination of predictors. One of the difficulties of Budescu's original approach is that dominance may not always be established between each and every pair of predictors.

The correlation measure previously reviewed, $R^2_{Y\cdot Xi}$, measures the relative importance of each variable alone. One of the advantages of using the DA process instead of simply using this measure is that DA measures relative importance in

a pair-wise fashion. As mentioned before, it also allows for this pair-wise comparison in the context of all models, both full and subset. Due to these advantages, DA is much more sensitive to any of the various importance patterns that one may see in the case of correlated and uncorrelated variables. It also addresses a more general importance definition.

Single Voxel. The first step in the DA approach is to build a correlation matrix. The size of this matrix is determined by the number of variables that are included in the analysis.

TABLE I
SAMPLE CORRELATION MATRIX

	Y	V1M	Edyrs	V1LT	V1LTr	Age
Y	1.000	0.208	-0.296	-0.106	0.158	0.048
V1M	0.208	1.000	0.205	0.205	0.354	-0.228
Edyrs	-0.296	0.205	1.000	0.363	-0.018	-0.186
V1LT	-0.106	0.205	0.363	1.000	0.702	-0.663
V1LTr	0.158	0.354	-0.018	0.702	1.000	-0.413
Age	0.048	-0.228	-0.186	-0.663	-0.413	1.000

A sample correlation matrix used in the analysis presented

The matrix should be set up so that the correlation values between each variable are recorded in the entry corresponding to the row determined by the first variable, whether outcome or predictor, and the column determined by the second variable. We also place this same value in the entry for the row determined by the second variable and the column determined by the first. We do this for all possible pair-wise comparisons. We can see that any row and column that line up with the same variables gives us a correlation coefficient equal to 1. Each of the other entries shows the correlation between each of the predictor variables. The other point to notice is that each (*i*,*j*) entry corresponds to each (*j*,*i*) entry as previously mentioned; the table is perfectly symmetrical.

Once we have developed a correlation matrix, we use it to calculate the squared correlation coefficients for the null, full, and all subset models. The squared correlations for the models containing only one variable can be found by squaring the coefficients found in the entries in the row representing the outcome variable and the columns representing the respective predictor variables. For instance, if the correlation between Y and X_i is 0.6, then $R^2_{Y \cdot Xi}$ is going to be equal to 0.6^2, or 0.36. This works for all single predictor variable cases. If there is more than one variable, we find the sum of squared error for the total model and the sum of squared error for the residuals in each respective model. We then use Equation 1 to calculate the r-squared values for each subset model containing more than one predictor variable. We repeat this until we have calculated all $(2^p\text{-}1)$ r-squared values.

We use these r-squared values to fill in an additional contribution table. The first column of this table contains all the possible subset models grouped by the number of predictor variables included in the model. The second column of this table contains the r-squared values in the same rows as their respective models. The next columns depend on the number of possible predictor variables. These columns form

TABLE II
R-SQUARED VALUES FOR ALL POSSIBLE MODELS

Model	R^2	Model	R^2
V1M	0.0433	V1M+Edyrs+V1LT	0.1754
Edyrs	0.0876	V1M+Edyrs+V1LTr	0.2245
V1LT	0.0112	V1M+Edyrs+Age	0.1667
V1LTr	0.0250	V1M+V1LT+V1LTr	0.0576
Age	0.0023	V1M+V1LT+Age	0.0497
V1M+Edyrs	0.1629	V1M+V1LTr+Age	0.0795
V1M+V1LT	0.0469	Edyrs+V1LT+V1LTr	0.1696
V1M+V1LTr	0.0572	Edyrs+V1LT+Age	0.1154
V1M+Age	0.0440	Edyrs+V1LTr+Age	0.2406
Edyrs+V1LT	0.1145	V1LT+V1LTr+Age	0.0328
Edyrs+V1LTr	0.1688	V1M+Edyrs+V1LT+V1LTr	0.2282
Edyrs+Age	0.0894	V1M+Edyrs+V1LT+Age	0.1884
V1LT+V1LTr	0.0250	V1M+Edyrs+V1LTr+Age	0.3807
V1LT+Age	0.0113	V1M+V1LT+V1LTr+Age	0.0813
V1LTr+Age	0.0328	Edyrs+V1LT+V1LTr+Age	0.2452
		V1M+V1LT+V1LTr+Edyrs+Age	0.4037

R-Squared Values for All Possible Subsets of Predictors

an additional contribution matrix for all possible subset models and predictor variables. If we have k possible predictor variables, then the first row of these columns represents the null model. The values under each variable equal the r-squared values of the respective models containing one variable. Once we have the r-squared values for each possible subset, we are able to fill in the matrix using an algorithm based on the difference in values of a "full" and "reduced" model. This means that we find the difference between proportions for the model containing j predictors and the model that contains $(j\text{-}1)$ predictors. For instance, if we want to calculate the additional contribution of adding X_2 to the model containing X_1, we have to calculate the difference

$$R^2_{Y \cdot X_1 X_2} - R^2_{Y \cdot X_1}$$

The calculated value is then placed in the row corresponding to the model containing only X_1 and the column corresponding to X_2. This process continues until we have filled in all the entries of the matrix.

After filling in the matrices for each group, we must also calculate the average contributions of each variable per group of subset models (i.e., model sizes). To calculate the averages of each variable per group, we simply sum all additive contributions for the first variable in Group One and then divide it by the number of non-zero entries. Once we have done this for the first variable and group, we repeat it for the second through the k^{th} variable in the first group. After the first group average has been calculated, we go to the second model size group and repeat the same algorithm and process. This is done up through the model of size $(j\text{-}1)$. This is simply due to the fact that the full model cannot have any additional contribution and so the average of all values must be equal to

zero. Once all of the subset average additive contributions have been calculated, we must also calculate an overall average additive contribution value for each possible variable. This is done by taking the sum of all average contributions and dividing by the total number of average values.

TABLE III
ADDITIONAL CONTRIBUTION MATRIX

Model	V1M [X1]	Edyrs. [X2]	V1LT [X3]	V1LTr [X4]	Age [X5]
Null	0.043	0.088	0.001	0.225	2.3e-03
Avg.	0.043	0.088	0.001	0.225	2.3e-03
			K=1		
X1	NA	0.120	3.6e-03	0.014	7.5e-04
X2	0.075	NA	0.003	0.081	1.8e-03
X3	0.036	0.103	NA	0.014	2.1e-05
X4	0.032	0.144	2.8e-06	NA	7.8e-03
X5	0.042	0.087	9.0e-03	0.030	NA
Avg.	0.0462	0.1135	0.0099	0.0348	0.0026
			K=2		
X1X2	NA	NA	0.013	0.062	3.8e-03
X1X3	NA	0.129	NA	0.011	2.8e-03
X1X4	NA	0.167	4.4e-04	NA	0.022
X1X5	NA	0.123	5.7e-03	0.035	NA
X2X3	0.061	NA	NA	0.055	8.8e-04
X2X4	0.056	NA	7.9e-04	NA	0.072
X2X5	0.077	NA	0.026	0.151	NA
X3X4	0.033	0.145	NA	NA	7.9e-03
X3X5	0.038	0.104	NA	0.022	NA
X4X5	0.047	0.208	5.0e-05	NA	NA
Avg.	0.0520	0.1459	0.0076	0.0559	0.0183
			K=3		
X1X2X3	NA	NA	NA	0.053	0.013
X1X2X4	NA	NA	3.8e-03	NA	0.156
X1X2X5	NA	NA	0.022	0.214	NA
X1X3X4	NA	0.171	NA	NA	0.024
X1X3X5	NA	0.139	NA	0.032	NA
X1X4X5	NA	0.301	1.8e-03	NA	NA
X2X3X4	0.059	NA	NA	NA	0.076
X2X3X5	0.073	NA	NA	0.130	NA
X2X4X5	0.140	NA	4.5e-03	NA	NA
X3X4X5	0.049	0.212	NA	NA	NA
Avg.	0.0801	0.2057	0.0080	0.1070	0.0671
			K=4		
X1X2X3X4	NA	NA	NA	NA	0.175
X1X2X3X5	NA	NA	NA	0.215	NA
X1X2X4X5	NA	NA	0.023	NA	NA
X1X3X4X5	NA	0.322	NA	NA	NA
X2X3X4X5	0.159	NA	NA	NA	NA
Avg.	0.159	0.322	0.023	0.215	0.175
Overall Average	**0.0760**	**0.1750**	**0.0119**	**0.0876**	**0.0531**

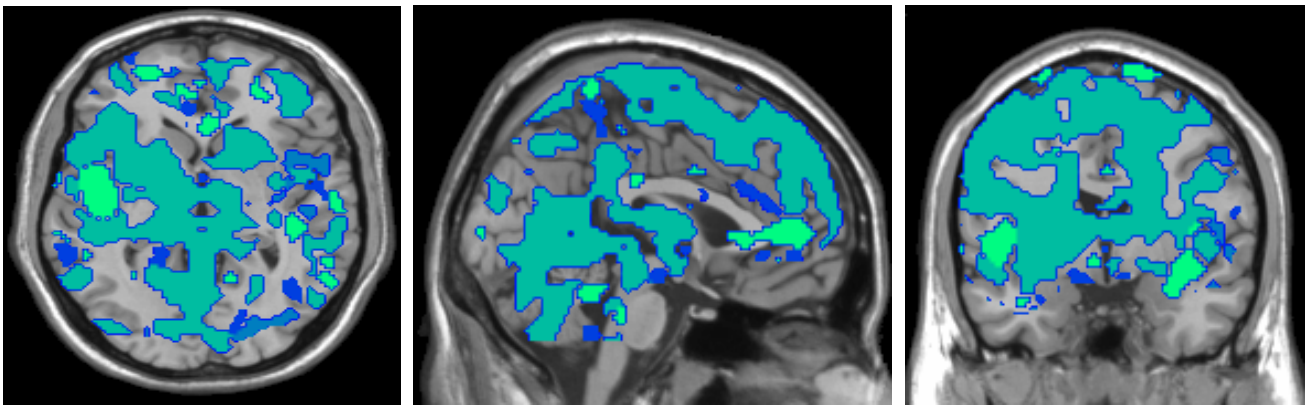
The additional contribution in R-Squared values in adding each covariate to a model.

For example, we can find the value for the model containing solely *V1M* by squaring its correlation coefficient with outcome *Y*. Therefore, $R^2_{Y \cdot V1M}$ is equal to $(0.208)^2$ or .0433. We can continue this process for all models. Using these values, we can then calculate the additional contribution matrix for the example. For instance, the additional contribution of adding years of education to the model containing the mini mental state examination is equal to $R^2_{Y \cdot V1M+Edyrs} - R^2_{Y \cdot V1M}$. Therefore, the additional contribution should equal (0.1630 - 0.0433) = 0.1197. Table 3 demonstrates this and all other additional contribution calculations in matrix form for this sample analysis.

Now that we have the additive contribution matrix, we are able to fully apply the dominance analysis. In order to establish dominance, we must examine these additive contribution values for each predictor. There are three types of dominance that can be derived from the additive contribution table [6]. A predictor is determined to *completely dominate* another predictor variable if its respective additional contribution to *all* subset models that are comparable is greater than that of the other predictor variables. Saying all comparable models is to say all the comparisons that involve the respective variable. This complete dominance analysis can be performed on many comparisons. We can look at specific comparisons, such as dominance between X_1 and X_2, or we can look to determine dominance between X_1 and all other possible predictor variables.

Using the contribution table to determine if X_1 completely dominates X_2, we compare the third and fourth columns of the contribution table. If one of these predictor variables (say, X_2) is found to have larger additive contribution values at all possible comparisons, then we can say that X_2 completely dominates X_1. In the case of finding one variable that is completely dominant over all others, we start with the first row in the first row in the first group of models (null). We find which variable has the largest contribution. From there, we move on to the next group of models and compare the values in the rows in which the dominant variable from the null model comparison are non-zero. If this variable has the highest values in each row of this group, we move on to the next group of models. We continue this process until we have reached the group containing all *k* variables. If we have made it to this model, then we have determined that the compared variable is completely dominant over all the other predictor variables. If it is found that there is no such variable for all possible comparisons, then we conclude that complete dominance cannot be established in this case. This is to say that dominance is rendered undetermined.

In order to reduce the frequency and incidence of this undetermined dominance occurring, two other forms of dominance may be established. The first of these is known as *conditional dominance*. After discovering that there are no dominant variables across all comparisons, we now look at the average contribution levels by model size. We follow the same process as complete dominance, except we only compare the averages of subset models of a given model size. We start out with the null model and determine which variable has the largest contribution to the null model (r-squared of each single variable). After this, we move to the averages that we had calculated in our table for all models containing only one predictor variable. If the same variable dominates as in the null case, we move on to the average of the models containing two predictors. This continues again until the full model has been reached. If one variable has the largest contribution after comparing the averages of all model sizes, we can conclude that this variable conditionally dominates all the others.



a b c

Figure 3. Dominance Analysis overlay for example data set. Displayed are the Transverse (a), Sagittal (b), and Coronal views (c) of the analysis. In this case variable that most dominated the results was the Trail making test (V1LTr) and is displayed by the aqua-green color that dominates the display.

If it is found that there is no conditionally dominant variable, then we move on to the last level of dominance. *General dominance* is determined when a variable has a contribution greater than another after considering the average of all conditional values. In our table, this is the overall average value that was calculated and represented in the last row. Since general dominance is determined using an overall average, it will always be possible to establish a generally dominant variable. This works when comparing two single variables or all possible predictors.

These three levels of dominance are related to one another, and a hierarchy is formed. Complete dominance will always imply conditional dominance. So too, conditional dominance will always imply general dominance. However, for $j > 3$, the converse may not hold; that is, general dominance does not imply conditional dominance, and conditional does not necessarily imply complete dominance [6].

Using the results from Table 3, we can first look for general dominance. The predictor with the largest overall average contribution will generally dominate the others. In this case, it appears as though *V1LTr* generally dominates the other four variables in this model. If we were to look at the averages per different model size, we would also see that this variable is also conditionally dominant over the others. The last one to check is complete dominance. In each row where the column representing *V1LTr* does not contain *NA*, we can see that the value is greater than the contributional value of any other variable. Therefore, we can say that this variable completely dominates the others in this voxel of the brain.

Full Brain. The previous example was merely an example of dominance analysis applied to a single voxel of the brain. However, in neuroimaging, we can apply this approach to every voxel of the brain and then create an overlay image. Figures 3 shows, in three dimensions, an image of the brain with the example analysis applied to it.

The five different colors represent the different predictor variables used in the model. Each colored region represents a voxel that is completely dominated by that single variable. In the single-voxel example, the voxel being studied would have been shaded with the color representing the *V1LTr* variable.

TABLE IV
SUMMARY OF DOMINANCE

Variable	No Dom.	V1M	Edyrs	V1LT	V1LTr	Age
Total Voxels Completely Dominated	73457	8813	4000	2477	89736	9204
% of brain area	39.138	4.696	2.131	1.320	47.812	4.904

Table gives the number of voxels and proportion of the brain that had complete dominance for each predictor used in example analysis.

In these images, we see a lot of color on our functional overlay. According to Table 4, the RAVL Trail Making test [X_4] completely dominates 47.812% of the brain. We can also see that in just over 39% of the brain, none of the predictor variables were completely dominant over the others. If we were to include more predictors in our model, we may not see color over so vast an area. This is due to the fact that it is harder to find complete dominance when more variables are added to the model; all things being equal, the more variables present in the model, the less likely it is for any one variable to completely dominate the rest. Therefore, we can anticipate that the percentage of the brain that doesn't have a completely dominant variable will be higher than 39.138%. We could also imagine that the respective individual percentages dominated by each variable might also decline.

VII. CONCLUSION

Dominance analysis is a relatively new tool in the statistical world. The purpose of this paper, along with the attached *R* code, is to assist researchers by establishing this procedure as a useful mechanism for detecting results. By performing dominance analysis on each voxel of the brain, it is possible to map out which tests and predictors are dominant at certain areas of the brain. Ideally, this form of analysis will give researchers a clearer picture of how the brain is functioning during a given test. However, the possibilities for using this method are almost unlimited; this method, and the code to

implement it, may well be applicable in many other fields of research.

ACKNOWLEDGMENT

We would like to thank Dr. Timothy Hess for his assistance with this project. His programming skills and statistical knowledge were helpful in getting us to this point. We would also like to thank Dr. Sterling Johnson for his cooperation with our team and for a behind-the-scenes look at his laboratory. Without his data and research, we might never have finished and our project might never have been possible.

REFERENCES

[1] Alzheimer's Association. "Alzheimer's Facts and Figures." 3 April 2009. http://www.alz.org/alzheimers_disease_facts_figures.asp?gclid=CK3gh-v6lZoCFRINDQod7nyhMQ.

[2] Johnson, Sterling. "Functional Imaging in Early Detection of Alzheimer's Disease." University of Wisconsin-Madison Institute on Aging. 5 February 2009. http://aging.wisc.edu/research/affil.php?Ident=120.

[3] Office of Health Care Access. "Final Decision: Yale-New Haven Hospital." 22 December 2008. http://www.ct.gov/ohca/lib/ohca/condecisions/decisions_2009/08_31290_wvr.pdf.

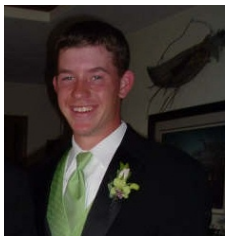
[4] Ashburner, J., K. Friston, and W. Penny. Human Brain Function. St. Louis, MO: Academic Press, 2003.

[5] Ridgway, Ged. "Spatial Preprocessing." Center for Medical Image Computing, University College, London. 2009. http://www.fil.ion.ucl.ac.uk/spm/course/slides09-zurich/.

[6] Azen, Razia. "The Dominance Analysis Approach for Comparing Predictors in Multiple Regression." American Psychological Association, Inc. Psychological Methods, 2003, Vol. 8, No. 2, pp. 129-148.

[7] Shieh, G. (2001). The inequality between the coefficient of determination and the sum of squared simple correlation coefficients. *The American Statistician*, 55, 121-124.

[8] Budescu, D. V. (1993). Dominance analysis: A new approach to the problem of relative importance of predictors in multiple regression. *Psychological Bulletin*, 114, 542-551.



Brett M. Wegner was born in Watertown, Wisconsin on September 24, 1986. He is currently finishing up his senior year at Ripon College in Ripon, Wisconsin with a Bachelor of Arts degree in mathematics and business administration with a minor in economics.

He has worked in the mathematics and computer science department at Ripon College for two years and in the athletic department for four. Following graduation, he will be attending the University of Wisconsin-Madison to earn a masters degree in accounting.

Mr. Wegner is a member of the Ripon College Laurel Society, Phi Beta Kappa, and Omicron Delta Epsilon. He currently plays on the Ripon Redhawk baseball team and participates in many intramural sports. He enjoys numbers, golfing, euchre, and being active.

Nicholas R. Krueger was born in El Paso, TX on 25 July 1987 and currently resides in Menomonie, WI. He is finishing his bachelor's degree at Ripon College in Ripon, WI in May 2009 with majors in mathematics and political science and with minors in Latin and national security studies.

He has been employed while attending Ripon College as a political science departmental assistant; as the student coordinator of the National Security Studies program; and as a mathematics tutor. In August 2009, he will enroll in Georgetown University's Security Studies Program to earn a masters degree in international relations.

Mr. Krueger is a member of the Ripon College Laurel Society and Phi Beta Kappa. He participates frequently in intramural sports and edits the Ripon College International Relations Review. He enjoys reading, playing softball, and enjoying the outdoors.

APPENDIX A: SAMPLE DATA

The example explained in the text was composed of five unique predictor variables that the medical research team in Madison provided us with. A small portion of the experimental data is presented below.

V1	V1M	Edyrs	Age	V1LT	V1LTr
2013	30	12	900	29	5
2023	27	16	875	31	0
2068	27	18	919	35	5
2073	24	12	925	25	2
2089	26	12	1,024	24	2

SUPPLEMENTARY MATERIAL

The R code for performing the Dominance Analysis Approach can be found at
http://ripon.edu/academics/macs/summation/2009/supp/dom-R.pdf

Summation
May 2009, pp. 15-18
http://ripon.edu/macs/summation

A Brief Introduction to Data Compression and Information Theory

Arthur H. Rumpf IV
Department of Mathematics and Computer Science – Ripon College

***Abstract*—With the current reliance on telecommunications, a way to shrink data and save on bandwidth is essential. There are myriad ways of attempting to reduce the space data occupies through applied information theory, including both lossy and lossless methods. This paper will examine several of those methods and the overall theory itself, with a focus on Kolmogorov complexity K(s).**

***Key Words*—compression, information, Kolmogorov, lossless, lossy.**

I. Introduction

DATA compression is now a nearly ubiquitous occurrence in modern computing. Just about every image on every page of the Internet is compressed into either the JPG or GIF format, many documents are compressed to save space, even most programs are compressed into installers to minimize possible download time. With the sheer abundance of compression in the modern world, it's important to have at least a cursory understanding of the factors and algorithms at work behind the scenes of these space-saving techniques.

To begin the examination of data compression methods and algorithms it is useful to have a working definition of compression itself. Compression is the act of taking a set of data and applying various algorithms to reduce the overall size of the data, while ensuring that the original set can be reproduced in part or in full at a later point. This definition grew out of the field of information theory, a method of dealing with information and the communication thereof. In his seminal 1948 paper "A Mathematical Theory of Communication", Claude E. Shannon created information theory when he observed that "the fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point." [1].

The two definitions given clearly have heavy similarities, perhaps most notably the similarity between "in part or in full" and the phrase "exactly or approximately". These lines are applicable to lossy and lossless compression methods.

Work presented to the college May 7, 2009.

A. H. Rumpf IV hopes to work at Gaming Informatics in Madison, WI 53713 USA (email: artrumpf@gmail.com).

A. *Lossy compression*

Lossy compression occurs when the original data set cannot be fully constructed from the compressed data set. As an example, suppose the datum given is the number 19.88888888. One lossy method of compressing this number would be to round to the nearest integer, in this case 20. On the one hand, the datum is shortened from 11 digits to two, but on the other some of the integrity of the original data is lost. It is impossible, once a number is compressed by rounding (or any other lossy compression method,) to flawlessly replicate the original data from the compressed set.

B. *Lossless compression*

As the name implies, lossless compression allows for the total recall of the original data set after compression. In the previous example, if one wanted to institute lossless compression instead of lossy, the most straightforward way would be to exploit patterns in the original data set. With the datum 19.88888888, the space after the decimal place is just the digit 8 repeated eight times, so it is easy to rewrite the number as 19.8*8 or in some similar fashion that implies the repetition of that digit. In this example, the number is only compressed to six digits from the original 11, illustrating the tradeoff between the efficiency of lossy compression and the fidelity of lossless.

II. Information Theory

As stated earlier, the field of data compression grew out of the study of information theory pioneered by Claude E. Shannon. Information theory deals with the quantification of information. "Information", in this case, refers to the most raw, compressed form possible – a thousand page encyclopedia has less information than a thousand page book of random characters, as the information in the encyclopedia could be compressed using the patterns of natural language and retain fidelity while there is very little that could be done to compress the book of random characters.

A. *Source and channel coding*

Information theory is interesting in that two of its main tenets – source coding and channel coding – can be observed in natural language quite easily. Source coding indicates that a communication system should be efficient: common items

should be the shortest, while more specific and uncommon items may be longer. This is apparent in English, where common words like pronouns are generally very short, while less common words like specific nouns are longer.

The other main tenet of information theory, channel coding, deals with the robustness of the communications language itself. In a properly channel-coded language, one or two of the words from the overall message should be able to be lost but be replaceable by context. For an English example, the sentence "The school was closed for a week over concerns about swine flu," retains most of its meaning even if a few of the words are incomprehensible or lost altogether due to the natural patterns in language. For another common-day example of channel coding, the robustness of compact disc encoding methods allows CDs to be played back even in the presence of minor scratches on the disc's read surface.

B. Kolmogorov complexity and randomness

Of course, with the unique definition and usage of the word "information" in information theory, one must find a way to quantify information. This leads to Kolmogorov complexity. The idea of Kolmogorov complexity was actually put forth by Ray Solomonoff in 1960, but it was independently discovered by Andrey Kolmogorov and Gregory Chaitin shortly afterwards. These three men further developed information theory and created the field of algorithmic information theory, dealing with the computation of quantified information. Despite Kolmogorov ceding his discovery claims to complexity to Solomonoff, the idea still bears his name due to popularity.

Kolmogorov complexity is, simply put, the amount of resources it takes to reproduce a given data set at a later point. Given the two strings, "ttttttttttttttttttt" and "a*J^fsj12#zdSWads$4P", both with a length of 20 characters, the first one will have a much lower Kolmogorov complexity as it can be rewritten in some manner of "t 20 times" which retains the same information in 10 characters. The second string, like the book of random characters in the example above, will likely have a Kolmogorov complexity of 20 as it is entirely random information with very few (if any) patterns that can be exploited to compress the information. It is important to note that Kolmogorov complexity only applies within a fixed language: while it is technically possible to create a symbol that means "reproduce the string 'a*J^fsj12#zdSWads$4P'", that symbol would need to be defined within the language itself and thus the end result would be longer than the original string.

This observation that many strings exist for which no program can reproduce them with less resources than the original string used itself is the basis for Kolmogorov randomness. A string is Kolmogorovically random if it is shorter than any program that can generate it. Most short strings fall into this category, since as the length of the string decreases the probabilities of patterns occurring in the string also decreases.

C. Incomputability of Kolmogorov complexity

One main issue present within the field of information theory is the incomputability of the Kolmogorov complexity of a string. If it were possible to compute the Kolmogorov complexity of a string, it would be possible to recreate some strings with fewer resources than the calculated complexity. I will provide a proof for this in short order, but before I do it may be helpful to bring up the Berry paradox. The Berry paradox, first discussed in print by Bertrand Russell, asks the reader to find "the smallest positive integer not definable in under eleven words." Since there is a large but finite number of words, there is also a large but finite number of phrases comprised of fewer than eleven words. As there is an infinite number of positive integers, at some point there stops existing corresponding phrases of less than eleven words. Once this happens, however, that number could be described as "the smallest positive integer not definable in under eleven words", which is itself only ten words. The expression is thus self-contradictory, as any integer that it defines is clearly definable in ten words, and therefore not defined by the statement at all [2].

This manner of self-referential paradox also arises with the theoretical computation of Kolmogorov complexity. This can be proven through contradiction. Assume that the Kolmogorov complexity of a string, K(s), is computable by the program ComputeComplexity. ComputeComplexity is a program with a length of U that takes in a string s and returns the integer K(s). Using this program, it would be possible to create another function called GenerateString of length C that takes in an integer n and generates every string starting with the shortest and continues until it finds a string s (using ComputeComplexity) such that K(s) is greater than or equal to n.

The problems arise here. The total length of the program combining ComputeComplexity and GenerateString is $U + C$. The input, n, is stored as a binary integer on the machine and thus uses $log_2(n)$ bits of storage. Since n, an integer, grows more quickly than $\log_2(n)$, it is possible to choose an n such that $U + C + log_2(n) < n$. This, of course, contradicts the requirement of having a complexity of at least n: by the definition of K(s), s should only be reproducible by a program at least n in length. Therefore, the assumption that K(s) is computable falls flat [3].

This is a significant proof that ties together the concepts in information theory and data compression. As the Kolmogorov complexity of any given string is incomputable, it is impossible to prove that a given string is compressed as efficiently as possible or if the string is Kolmogorovically random. It is, of course, still possible to compute the upper bound of a string's Kolmogorov complexity – in the example that could be compressed as "t 20 times", the complexity is at most 10 characters.

III. Data Compression

With the requisite introduction to information theory taken care of, it is possible to look more in depth at data compression itself. The two types of compression, as stated before, are lossy and lossless. Items that are lossily compressed permanently lose some of the original information, while lossless compression methods retain everything. The downside to lossless compression is that most of the time, a lossy compression format will create a significantly smaller compressed file than any known lossless method could produce. Lossy compression is best suited for situations where not all information must be present, such as audio files, video files, and pictures. When compressed in a lossy format, the amount of information lost is generally acceptable in these media – even if a picture of a house appears grainy or some color information has been lost, the average viewer would still be able to recognize the image as of a house. This phenomenon where the brain fills in missing information to get a message makes lossy compression viable. Common lossy compression methods include .jpg and .gif images, .mp3 audio files, and MPEG-4 video files.

Lossless compression, on the other hand, is desirable when fidelity is an absolute must. This is the case with program installers, text, archived files, and the like. After all, it would be terrible to download an eBook and have parts of the chapter missing, or to open a program and find that it has lost vital information. Common lossless compression methods include run-length encoding, dictionary encoding, and byte-pair encoding.

A. Run-length encoding

Run-length encoding is the simplest form of lossless data compression available – in fact, the example given in the introduction where the data was shrunk by acknowledging repeated digits was run-length encoding. Simply put, run-length encoding takes runs of data – strings or sequences where data is repeated – and replaces them with a key allowing the repetition to be replicated. Run length encoding is used heavily in fax machines, along with other types of encoding, as most faxes are mostly white with patches of black.

B. Byte-pair encoding

Byte pair encoding is a form of encoding in which the most commonly occurring pair of bytes is replaced with a single byte that does not appear in the source file. Of course, with this method the compression algorithm must maintain a table with a key to decode the compressed file.

As an example, say the data to be encoded is “eeeteeeteb.” The encoder would find that the most commonly occurring pair of bytes is “ee”, occurring a total of four times. The encoder would then go through the data and replace every occurrence of “ee” with a byte that doesn’t occur in the original data – for this case we will say it will be replaced with “Z”. The encoder creates a table that indicates that “Z = ee” and reduces the original data to “ZetZeteb”.

One of the main advantages to byte pair encoding is that it is easily repeatable. On another run with the same data, the encoder will find that “Ze” is the most commonly occurring pair. It will find another byte not found in the data and replace each occurrence of “Ze” with it. For the sake of example the encoder will choose “A” as the new code. So after the second run, the data becomes “AtAteb” and the table has the lines “Z = ee” and “A = Ze”.

Of course, that’s not the end of the line. Another run through the data shows that the byte pair “At” occurs more than once, and can thus be replaced. The byte pair encoder can choose another byte that doesn’t appear in the source, say “T”, and update the reduced data and table accordingly to “TTeb” and “Z = ee, “A = Ze, T = At”. At this point, no pair of bytes occurs more than once in the data and the compression is complete.

In the example, it is important to note that since the original data set was so small, there were not many repetitions of any particular byte pair. Thus, the resultant compressed code, when combined with the key table, is larger than the original code. With a larger data set, however, the compression ratio can be significant.

C. Dictionary encoding

Dictionary encoding is a more generalized form of byte-pair encoding. Dictionary encoding searches out and replaces entire runs of commonly occurring data with a smaller string maintained in the key table or “dictionary.” In some cases, the coder may start with a standard dictionary. This is often the case for situations like large, fixed files (like books) are stored in a heavily limited media (like a cell phone or PDA). In other cases, the dictionary may be initialized with a few common items, but allow the contents of the dictionary to change during compression (much like the key table in byte pair encoding). This type of encoding is an extremely common form of lossless compression – the popular ZIP format of compression uses Deflate as its compression algorithm, which is essentially a modified form of dictionary encoding.

IV. Conclusion

In a world where the size and complexity of files continues to grow and evolve while the infrastructure in place for data transfer is often decades old, the importance of efficient compression techniques is absolutely essential. By using both lossy and lossless methods of compression where appropriate, it is possible to save time, bandwidth, and money.

Acknowledgment

Thank you to my group members, Dan Syens, Colin Harter, Curtis Schmitt, and our advisor Professor Kristine Peters. Also thanks to the rest of the math and computer science department for all of the help, inspiration and support through my years here at Ripon College.

REFERENCES

[1] C. E. Shannon, "A Mathematical Theory of Communication," in *Bell System Technical Journal*, vol. 27, July, October 1948, pp. 379–423, 623–656.

[2] G. J. Chaitin, "The Berry Paradox". IBM Research Division. Physics-Computer Science Colloquium. University of New Mexico. Albuquerque, 27 October 1993.

[3] P. B. Miltersen, "Course notes for Data Compression – 2: Kolmogorov complexity," 29 September, 2005.

[4] A. Feldspar. (23 August 1997). An Explanation of the Deflate Algorithm. [Online]. Available: http://zlib.net/feldspar.html

[5] N. Abramson, *Information Theory and Coding*. New York, New York: McGraw Hill Book Company, 1963, pp. 1–92.

[6] G. E. Blelloch, "Introduction to Data Compression". Carnegie Mellon University. 16 October, 2001.

[7] M. Nelson. (20 June 2006). The Million Random Digit Challenge Revisited. [Online]. Available: http://marknelson.us/2006/06/20/million-digit-challenge/

[8] T. Schneider. (9 Febuary 1999). Information Theory Primer. [Online]. Available: http://www.ccrnp.ncifcrf.gov/~toms/paper/primer/

Arthur H. Rumpf IV was born in Oconomowoc, WI in 1987 and studied at Oconomowoc High School before coming to Ripon College.

He has worked at the Medical College of Wisconsin as a summer intern to the Bioinformatics department, doing programming work for the Rat Genome Database.

Mr. Rumpf is, in addition to computing, interested in musical composition and the science behind game design. He is eagerly awaiting the summer, when he can resume his hobby as a windsurfer.

Summation
May 2009, pp. 19-24
http://ripon.edu/macs/summation

Lossless Image Compression

Curtis J. Schmitt
Department of Mathematics and Computer Science – Ripon College

***Abstract*—Image compression is becoming an ever growing field with the need to save space while storing or transmitting digital images. Lossless compression uses techniques that allow the data to be compressed so that when uncompressed the original data can be reconstructed. In this paper I will discuss two methods of lossless compression, differential pulse-code modulation (DPCM) and arithmetic coding. I will also show how they can be used together to achieve even better compression.**

***Key Words*—Arithmetic coding, DPCM, image compression, lossless**

I. Introduction

IMAGE compression's purpose is to reduce the amount of the data being stored. This can be accomplished in one of two ways, lossy compression and lossless compression. Each has its place in the world of digital imagery. In the process of lossy compression information is lost during the quantization step. Quantization represents all values in a certain range as a single value. In the case of an image this would generally mean using fewer colors. To accomplish this many similar colors would all become a single, different color (Figure 1).

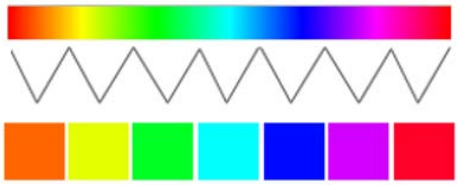


Figure 1. Quantization of a color space. The large color space on top is quantized by a assigning all of the colors in the given range to a single color, below.

Because information is lost the original data can never be exactly reproduced. By essentially throwing out information a high compression ration can be achieved. This is good when small file sizes are desired, such as for use on the internet. But when detail is essential, lossless compression techniques are used. For example, in medical imaging even a small change in the image could indicate the difference between a serious medical issue and a clean bill of health. Also, archival images would benefit from being compressed using a lossless technique.

Work presented to the college April 10, 2009 as the forth paper in a series of papers dealing with data and image compression techniques.

C. J. Schmitt plans to enter the computer software industry as a developer after graduation from Ripon College, Ripon, WI 54971 USA.

Differential pulse-code modulation (DPCM) uses prediction to estimate the colors in the image. This method of compression then finds the difference between the predicted values and the actual values and stores these differences [10]. We will discuss this process later in greater detail. DPCM is not very widely used in image compression, in favor of less computationally intense methods of lossless compression. Because it is not very widely used I found the concept of DPCM for image compression compelling and worth consideration.

Arithmetic coding is a rather popular method of lossless compression but again it is not widely used for image compression specifically. In this compression algorithm, all of the image data is used to calculate a single real number between 0 and 1. This number along with a table, created during the encoding process, is stored. By using the table and this number all of the original image data can be reconstructed with no loss of information.

Both DPCM and arithmetic coding are more computationally intensive than most other lossless compression methods and this is likely the reason that they are not very widely used. However, they do tend to produce very good compression ratios and in the case of arithmetic coding, near optimal results [8]. Further, when used together we will see how DPCM can complement arithmetic coding to achieve even better compression ratios.

II. Digital Image Basics

Digital images consist of two main parts, a header block and the pixel information. A pixel is the smallest unit of information in an image. Each pixel contains information about the color at a certain spot in the image. These pixels are then arranged in a row/column fashion to display an image [7]. The header is different for each image file type and is standardized for each. It will include information like the size of the image stored as a height and width given in pixels. Some sort of identifier also needs to be stored indicating what the image type is, if it is compressed, and what type of compression was used. The pixel information is then stored based on the image standardization, determined for each image type, and the type of compression used. In order for this file to be read the image viewer will also need to know how to interpret this information. This is again determined by the standardization of the image and must be properly implemented in the image viewer for each separate image type to be properly displayed.

The most common way to store a pixel's color information is in the RGB format. This breaks down the color into red,

green and blue components. Each of these color values is stored in 8 bits or a byte making each pixel 24 bits [7]. This makes representing over 16 million different colors possible. However, for the purposes of this paper we will be using grey-scale images. This means we will store just one 8 bit value for each pixel. This will give us 256 different shades of grey that we can use to represent each pixel.

III. Differential Pulse-Code Modulation

A. Pulse-Code Modulation

During pulse-code modulation the continuous analog signal is sampled at a certain interval (Figure 2), how often the samples are taken is measured as the sample rate or bit rate. At these intervals the amplitude of the signal is measured. These samples are then translated into a discrete-time signal which is stored digitally. The higher the bit rate the more closely the original data can be recreated. Also, the higher the bit rate the more data will need to be stored.

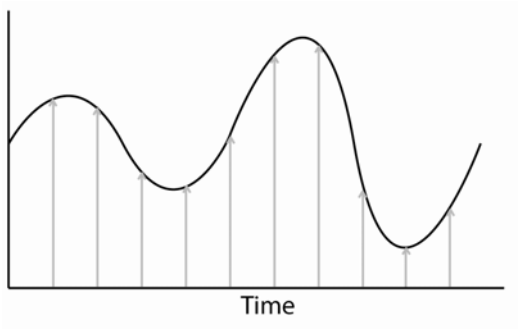


Figure 2. Sampling an analog, continuous-time signal.

It is evident that this is a lossy step and is actually a type of quantization. However, for my purposes we will not need this step because almost all images are captured and stored digitally. The imaging devices, such as cameras and scanners, will do this step for us and we can concentrate on the lossless step of the process.

B. The DPCM Process

We start the DPCM process with our input, a digital image. For simplicity we will be using a grey-scale image, an image with no color only 256 shades of grey. It should be noted that if we start with a color image and convert it to a grey-scale image we will be reducing the color space, a lossy step. We are not concerned about the information in the header block as it will need to stay intact. We will be dealing with the pixel data which can most easily be worked with in a zero based, 2-dimensional array. This array will store the shade of grey for the corresponding pixel in the image. That is, the pixel in the upper-left corner of the image will be stored in position (0,0) in the array.

1) Encoding

For the encoding example we will use a 5 pixel by 5 pixel image. A representation of this image can be seen in Figure 3. This image will then be processed to generate a 2-dimensional array of values representing the values of the pixels.

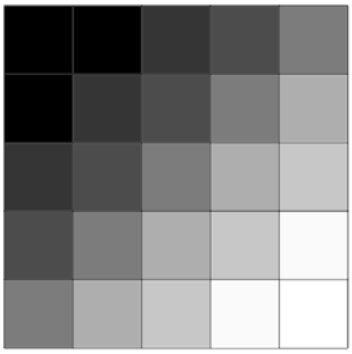


Figure 3. Grey-scale gradient image. This representation of a 5 pixel by 5 pixel is shown here, where each square represents a pixel. The image is a gradient of black, in the upper left, to white, in the lower right.

Using this 2-dimensional array (Figure 4a) we can begin the encoding process. This process involves, first, predicting pixel values, finding the errors in those predictions and then storing those errors. The idea being that the errors will be smaller than the actual values of the pixels, thus reducing the amount of information that will need to be stored.

We will start by predicting the pixel values in our image by taking the weighted average of neighboring pixels. We can choose any number of pixels and in any location, as long as they are above the given pixel or in the same row but to the left. It is possible to evaluate the image before the predicting step to decide what would be the best prediction formula. We can express this formula in the form

$$P(x,y) = w_1 p_1 + w_2 p_2 + ... + w_n p_n . \quad (1)$$

Where p_x is a previous pixel and w_x is the weight on that pixel. The weights will add up to one but do not have to be equal. This makes sense because it is likely that a pixel directly above or to the left will be closer to the same value than a pixel at a diagonal.

For our example we will choose the pixels immediately to the left, A, immediately above, C, and the pixel above and to the left of the predicted pixel, B.

B	C
A	X

Figure 5. Prediction method used in this DPCM encoding process.

We then assign these pixels weights, giving us

$$X = .35A + .30B + .35C . \quad (2)$$

As I mentioned before the weights can be anything as long as they add up to one, as these weights do [10]. Another important note is that because of this chosen equation for prediction we will not be able to predict all of the values in the image. We cannot calculate X for any pixel that does not have a pixel above or to the left of it. Therefore, we must store the actual values of the leftmost column and topmost row in the image. From here we can start calculating the predicted values starting at position (1,1). We use A, B and C from the original image and store all of the predicted values in another 2-dimensional array (Figure 4b). Now we take the difference of the corresponding positions in the original pixel data and the predicted values. This gives us the 2-dimensional array of prediction errors (Figure 4c) and we will store this.

	0	1	2	3	4
0	0	0	50	75	125
1	0	50	75	125	175
2	50	75	125	175	200
3	75	125	175	200	250
4	125	175	200	250	255

(a)

	0	1	2	3	4
0	0	0	50	75	125
1	0	0	35	68	110
2	50	35	68	110	160
3	75	68	110	160	193
4	125	110	160	193	235

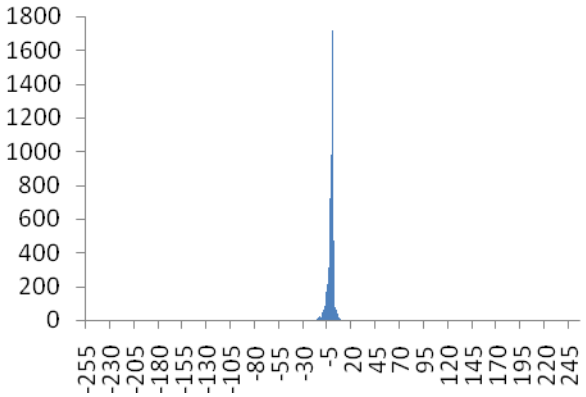
(b)

	0	1	2	3	4
0	0	0	50	75	125
1	0	50	40	58	65
2	50	40	58	65	40
3	75	58	65	40	58
4	125	65	40	58	20

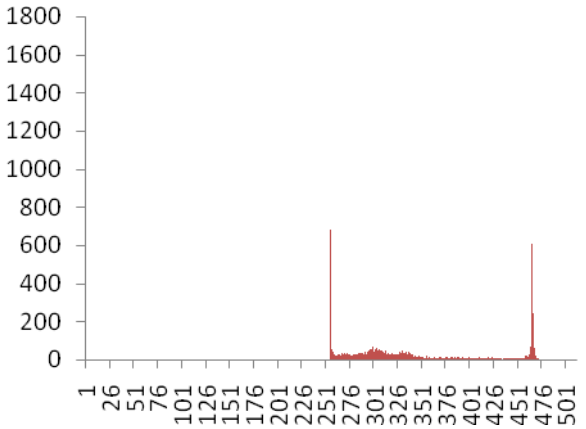
(c)

Figure 4. 2-dimensional arrays of values representing Figure 3. (a) shows the original values of the pixels. (b) shows the prediction errors found in the encoding process. (c) shows the difference in the actual values and predicted values.

After using the DPCM process we are left with the array of differences. If we look a frequency of all of the values (Figure 6a) we can see that the values are now more heavily centered on zero. If compare this to the histogram of the original values (Figure 6b) we can see the difference. There are simply more values with lower frequency [10]. This will have a positive impact later when we use arithmetic coding on this data.



(a)



(b)

Figure 6. Histograms of pixel values before and after DPCM encoding. Histogram (a) contains the values and their frequencies after the DPCM encoding of a 32 pixel by 32 pixel grey-scale image. Histogram (b) shows the values and their frequencies from the same image before encoding.

2) Decoding

The decoding process is simply the opposite of the operations of the encoding process. The only information that is needed for the decoding process is the prediction formula and the array of the prediction errors.

We start by creating a new 2-dimensional array and copying the leftmost column and topmost row from the array of prediction errors, remembering that we stored those as original values. Using this array we now begin to calculate the predictions starting at position [1,1], just as we did in the encoding process. Immediately after finding this prediction we add the prediction error back to this value, giving us the original pixel value. We continue this process, again, as we did in the encoding process by going left to right in each row

and top to bottom through the rows. This will give us the original array of data that we can now display with no lost information.

IV. Arithmetic coding

A. Premise

Arithmetic coding is a type of Huffman coding. In Huffman coding the goal is to find which symbols appear most frequently in the data. These symbols are then represented with fewer bits while less frequently occurring symbols are represented with more bits [1]. As you can see this should lead to less bits being needed to represent the image. However, this adds in overhead as the representations need to be linked back to the original symbols in some sort of dictionary.

Arithmetic coding is more complex and instead of coding each individual symbol in the data it encodes all of the data as a single fraction on the interval [0,1). Arithmetic coding still relies on the symbols' frequencies in the data, which will be seen in the encoding process.

For our specific case we will, again, be using a small grey scale image. However, because of the more complicated process we will be using a smaller five pixel by five pixel image. Our input will then be given in the same format as in the DPCM example (Section III.C).

B. Encoding

Given the 2-dimensional array representation of the image (Figure 7), we will start by processing the data.

	0	1	2
0	0	0	50
1	0	50	75
2	50	75	125

Figure 7. 2-dimensional array of pixel values. These values are taken from the 3 pixel by 3 pixel block from the upper left corner of Figure 3.

The first step is to construct the symbol table (Table I). This table will include each of the individual symbols in the data, their frequency and each symbols range, which we will calculate. We first go through the data, counting each of the symbols. When we are finished we will put them into the symbol table sorted by frequency, largest to smallest. Then we will calculate the range for each symbol, storing the low and high value. This is done by calculating the probability of that symbol showing up, frequency divided by the total of the frequency column, then adding this to the high value of the range from the last symbol [2].

TABLE I
SYMBOL TABLE

Symbol	Frequency	Low(l)	High(h)
0	3	0	1/3
50	3	1/3	2/3
75	2	2/3	8/9
125	1	8/9	1

Symbol table for arithmetic coding process. This table is created during the encoding process, stored and used during the decoding process.

Now that we have the symbol table we can start calculating the code number. Here, again, we will use a table for visualization purposes (Table II). In this table we will need to keep track of the symbol, the size of the range, the low value and the high value. These values will be explained as we go along. We start by listing out all of the symbols in the order that they appear in the data. We will also write down our starting high and low values, 0 and 1. Then we calculate our range for the first symbol which will be the high value minus the low value from the previous line. Next we will calculate the low and high values for the current symbol with the following equations:

$$L = L_p + R * l \tag{3}$$

$$H = L_p + R * h \tag{4}$$

where H and L are the high and low values, H_p and L_p are the previous high and low values, h and l are the high and low ends of the range from the symbol table, and R is the range in the current row. We continue calculating the range, then the high and low values for each symbol in the remaining rows. When we are finished with all of the symbols we have a final range. We can store any fraction in this range as our code number. Therefore, it makes the most sense to store the least precise value in that range [2]. In our case that would be 0.044465. We also need to store the symbol table. With these two pieces of information we will be able to reconstruct the original image without any lost data.

TABLE II
ARITHMETIC ENCODING CALCULATION RESULTS

Symbol	Range(R)	Low Value(L)	High Value(H)
		0.0000000000	1.0000000000
0	1.0000000000	0.0000000000	0.3333333333
0	0.3333333333	0.0000000000	0.1111111111
50	0.1111111111	0.0370370370	0.0740740741
0	0.0370370370	0.0370370370	0.0493827160
50	0.0123456790	0.0411522634	0.0452674897
75	0.0041152263	0.0438957476	0.0448102423
50	0.0009144947	0.0442005792	0.0445054108
75	0.0003048316	0.0444038002	0.0444715406
125	0.0000677404	0.0444640139	0.0444715406

Table of results from the arithmetic encoding process using (3), (4) and the process described.

It is important to see how our previous work with DPCM can help produce a smaller code number and symbol table, thus producing a better compression ratio. First, when DPCM is performed on an image we have seen that there is a fewer number of symbols with higher frequencies. This means that the symbol table, which must be stored in order to decode the data, will be smaller and therefore take up less space.

Furthermore, the smaller symbol table has an additional effect on the encoding process. By having fewer symbols, our ranges are also larger. Larger ranges mean that the high and low values that define those ranges are less precise, have fewer decimal places and require less space to be stored. This is important because going back to (3) and (4) we can see that multiplying by less precise fractions will give us less precise answers and finally a less precise code word. Again, this less precise code word will require fewer bits to store, which is our goal.

C. Decoding

As mentioned previously, the input for the decoding process is the code number and the symbol table that was constructed in the encoding process. To help with the visualization process we will, again, use a table (Table III) to show the progression of this process.

We start with the code number as our first "Number." Examining the symbol table we look to see which range this number falls into, this is our first symbol. We enter the symbol and the size of its range, from the symbol table, into the decoding table.

To find all of the remaining symbols we start by finding the next "Number" with the equation

$$N = \frac{(N_p - l_p)}{R_p} \tag{5}$$

where N is the "Number," N_p is the previous "Number," l_p is the low end of the range from the previous symbol, and R_p is the previous "Range." After calculating this value we will again find which symbol's range it falls into and that is the next symbol. Then range is then calculated and stored. This process repeats until we have all of the pixels from the original image [2]. We will know when we have them all because the header will carry that information.

TABLE III
ARITHMETIC DECODING CALCULATION RESULTS

Symbol	Range(R)	Low Value(L)
0	0.3333333333	0.0444650000
0	0.3333333333	0.1333950000
50	0.3333333333	0.4001850000
0	0.3333333333	0.2005550000
50	0.3333333333	0.6016650000
75	0.2222222222	0.8049950000
50	0.3333333333	0.6224775000
75	0.2222222222	0.8674325000
125	0.1111111111	0.9034462500

Table of results from the arithmetic decoding process using (5) and the process as described.

V. AN IMAGE COMPRESSION PROGRAM

A. Overview

A program was written to use these compression techniques in a practical way. It is written in C#.NET. The goal of the program was to use the DPCM method on an image and compare its compressed size to the uncompressed size, then further compress it with the arithmetic coding and compare its size. However, some limitations arose along the way and I only had limited success.

The DPCM process was successfully implemented in this program. Given an array of pixel values the program uses equation (2) to calculate the predictions and determine the errors. The errors are stored in a 2-dimensional array. This array is then processed with the arithmetic encoding techniques discussed in section IV, part B. However, at this step, because of coding techniques used there is a very large limitation on the image's size. This is due to the limited precision of the decimal variable used to store the code number.

When a small image is used the program is also able to successfully decode the code number, retrieving the DPCM encoded error array. The DPCM decoding process is then used to produce the original pixel data in a 2-dimensional array.

B. A Problem of Precision

The only major limitation of the program to date is the means used to calculate the code word in arithmetic encoding. As previously stated the arithmetic encoding process itself has been tested and is working. However, there is a limit to the size of the image. Very small images, 3 pixels by 3 pixels and 5 pixels by 5 pixels, have been used to test the encoding algorithm and the results were positive. The code numbers obtained were able to be decoded and the original images reconstructed and displayed. With images any larger than this the range obtained from the encoding process was too small to be stored in the decimal data type in C#, which is currently how the algorithm is being implemented. The decimal data type is 28 bits and is the largest data type that will store a decimal value. The problem occurs when the code number is being calculated. Once the high and low end of the range become so close together that there is not enough precision to store the difference.

The remedy to this problem is to not use the built in decimal data type. Instead we would have to use the byte data type and using bit manipulation we would have to store the code number in multiple byte variables, as many as would be needed for the required precision [5]. Due to time constraints this will not be implemented at this time.

C. Future Work

The first change to this program would be to implement the storing of the code number in byte data types rather than the decimal data types. I would also like to switch from grey-scale images to using 24 bit RGB images. This would allow for a more interesting program and would likely generate more interesting compression results.

In addition to those changes I would like to implement the saving of these compressed files. This would allow me to examine the realistic size of the files as they exist when saved to disc. I then hope to compare the ratios to the current, popular lossless file types that exist today.

ACKNOWLEDGMENT

I would like to thank Professor Kristine Peters and Professor David Scott for all of the mentoring, assistance and support that they have given me throughout my time at Ripon College. I would also like to thank Professor Peters for all of her assistance while writing this paper. In addition, I would like to thank the other members of my Senior Seminar group, Dan Syens, Colin Harter and Arthur Rumpf IV, for their support through this process.

REFERENCES

[1] Bodden, Eric, Malte Clasen, and Joachim Kneis. Arithmetic Coding revealed. Tech. no. 2007-5. 25 May 2007. Sable Research Group. 20 Apr. 2009 <www.sable.mcgill.ca>.

[2] Campos, Arturo. "Basic Arithmetic Coding." Arturo Campos home page. 22 July 1999. 06 Mar. 2009 <http://www.arturocampos.com/ac_arithmetic.html>.

[3] Hankerson, Darrel R. Introduction to Information Theory and Data Compression. Boca Raton, Fla: Chapman & Hall/CRC P, 2003.

[4] Hoggar, S. G. Mathematics of Digital Images Creation, Compression, Restoration, Recognition. New York: Cambridge UP, 2006.

[5] Howard, Paul F., and Jeffery S. Vitter. Practical Iplementations of Arithmetic Coding. Tech. no. 92-18. Norwell, MA: Kluwer Academic, 1992.

[6] MacKay, David J. C. Information Theory, Inference, and Learning Algorithms. Cambridge, UK: Cambridge UP, 2003.

[7] Markovitz, Barry, and Dana Braner. "Digital Image Compression: Pixels n' Bits." Journal of Intensive Care Medicine 18 (2003): 229-30.

[8] Null, Linda. Essentials of Computer Organization and Architecture. Sudbury, MA: Jones and Bartlett, 2006.

[9] Pracko, Richard, and Jaroslav Polec. "DPCM Application to Images Scanned by SFC Method." Journal of ELECTRICAL ENGINEERING 58 (2007): 161-64.

[10] Salomon, David. Data Compression: The Complete Reference. New York: Springer, 2006.

Curtis J. Schmitt was born in Beaver Dam, Wisconsin on November 17, 1986. He will be graduating from Ripon College of Ripon, Wisconsin United States. In May 2009 he will obtain a BA with a major in computer science and minors in mathematics and business administration.

Curtis has received practical experience in the software development industry working for WTS Paradigm as a Developer Intern. He will be obtaining a career in the software development industry in the greater metropolitan Detroit area in Michigan.

Mr. Schmitt is the former Chapter President of the Phi Delta Theta Fraternity. He enjoys fixing, working with and learning about the ever-changing world of technology and computers.

Summation
May 2009, pp. 25-31
http://ripon.edu/macs/summation

An Overview of JPEG Image Compression

Daniel S. Syens
Department of Mathematics and Computer Science – Ripon College

***Abstract*—In today's world, we interact with large quantities of data every day. Compressed data is useful for saving storage space and speeding up data transmissions. This paper examines a common image compression method called JPEG. It surfaces frequently on the web as well as being utilized by many digital cameras. I will examine the steps involved in the JPEG compression method including the most significant step, the discrete cosine transform.**

***Key Words*—Data compression, discrete cosine transform, image compression, JPEG**

I. INTRODUCTION

In an increasingly technological age, people are working with large quantities of data almost every day. Cell phone, PDA, computer, and Internet usage is commonplace for many. Billions of bytes of text and images are sent over the Internet every day, and from there they might be saved to hard drives, CDs, or flash memory. From this, one might conclude that it is useful to be able to take data and represent it in as small a form as possible. Thus, it takes up minimal space, and can be transmitted quickly. For example, if we had pieces of data each 1 megabyte in size, and a 10 megabyte storage device, then we could fit 10 instances of the data on our storage device. However, if we could compress them and obtain a size of 2.5 megabytes, we could fit twice as much data in the same space. Also, the smaller the data, the more information we can send back and forth in a given amount of time.

II. DATA COMPRESSION

Data compression really involves two ideas. First, there is the compression of the data, and second there is the decompression of the data. When data is compressed, the original information is stored in some alternative representation, which is ideally smaller than the original representation. Decompression is essentially the reverse, where the original information can hopefully be extracted from the alternate representation.

A simple text example can be used to demonstrate this concept. Let a string of text, *S*, equal "aaaabcdeaaaa." We will let this be the original, uncompressed form of *S*. Its length is twelve characters. Now, let's compress *S* into a new, alternate representation *T*, which is "4abcde4a." The length of *T* is only 8 characters. Clearly, the compressed form of *S* is smaller than the original.

To clarify how *T* and *S* represent the same information, let's decompress *T*. In *T*, we've simply represented redundant runs of the same character by placing a coefficient in front of that particular character. If there is no coefficient, then there is merely one instance of that character. Thus, "4a" is equivalent to "aaaa." So, if we were to decompress *T*, we would obtain the original information, "aaaacdeaaaa", with 100 percent accuracy. This is known as run-length encoding.

However, sometimes data is compressed into a form where 100 percent accurate retrieval of the original information is not possible. For a very simple example, consider the sentence "The apple is round and red." To compress this sentence, we will remove any instance of the letter *e*. The compressed sentence is "Th appl is round and rd." This sentence is most certainly shorter than the original, however, we have no real way of knowing how many or where an *e* has been removed. However, for many people accustomed to the English language, the sentence may still be completely readable, despite some lost information. The main idea is still retained.

III. LOSSY AND LOSSLESS

The two simple examples above help to illustrate a categorical division in data compression. If the original information can be completely and accurately reconstructed from the compressed version of the data, then the compression is considered lossless. However, if the reconstruction is merely a close approximation to the original data, then the compression is considered lossy.

In cases dealing with text, lossy compression is not generally utilized. With so many different words and possible letter combinations, there can be much ambiguity in the meaning of a reconstruction. However, with other kinds of data, loss can be quite acceptable.

For instance, both audio and images are often compressed using lossy algorithms. The information lost with these methods is often times unperceivable to the human eye and ear due to our own physical limitations. This will be examined further when discussing the JPEG compression method for images.

Work presented to the college April 29, 2009 as the third paper in a series of papers dealing with data and image compression techniques.

D. S. Syens plans to enter the workforce as a software developer after graduation from Ripon College, Ripon, WI 54971 USA (email: syensd@ripon.edu).

IV. Image Compression

In particular, this writing will examine a specific type of compression used when dealing with image data. This method is JPEG compression. Before examining the method, however, it is important to understand some general ideas about images.

An image is composed of hundreds or thousands or more dots, and each has a value associated with it. These dots are called pixels, and the value they hold represents visual information such as color value in a color image or an intensity level of black in a grayscale image. In general, image compression methods attempt to reduce the redundant information amidst the pixels in order to store images in the smallest form possible, while still retaining much, or all, of the visual detail.

However, just like data in general, there are many different types of images. The types are defined by the way color or value is distributed within the image. For example, a cartoon image is very different from a continuous tone image such as a photograph of a person's face. The cartoon image has lots of sharply defined areas of solid color, or many consecutive pixels with the exact same value. The photograph may contain many runs of pixels with similar values, but they are not exactly the same.

Let us think back to the example in section II, where we compressed data by adding coefficients in front of repeated data, rather than outputting multiple identical values. If we were to use this idea on the cartoon image, we could shrink down the image representation. That is, imagine that there a 400 x 50 section of blue sky in the cartoon image, and that blue sky is all one value of blue. The sky would be represented by the same value of blue in 20000 individually represented pixels. This can be easily compressed into a much smaller piece of information, i.e. "this blue repeated 20000 times" where one pixel is saved along with a coefficient to communicate how many times that pixel is repeated. However, that same idea doesn't work in the realistic photograph, where perhaps the blue sky has subtle different tones of darker and lighter blue interspersed throughout. Certainly, there are different algorithms better suited to different images.

This writing will be focusing on an algorithm typically used on continuous tone images. This is the JPEG compression method.

V. What is JPEG?

The JPEG compression method was developed in the early 1990s, receiving approval in September 1992. It was named for the group responsible for its creation, the Joint Photographic Experts Group. This group was formed as a collection of companies and academic groups who desired standards for still image compression [5].

As a compression method, JPEG is most useful when used with continuous tone images. These images typically have smooth color variation and are most often realistic photographs or paintings. They do not tend to be cartoon images, line art, or vector art. The reasoning will become clearer as the algorithm is explained.

VI. The Algorithm at a Glance

The JPEG method of compression and decompression of images is more or less symmetrical. That is, to decompress an image compressed using the JPEG algorithm, one simply reverses the steps. Of course, since it is a lossy algorithm, you do not necessarily get the exact same information back.

The compression algorithm can be broken down into a fairly simple set of steps. Typically, the color space is reduced, the image is broken up into smaller chunks, and each chunk is transformed using the discrete cosine transform, quantized, encoded, and saved.

Since the discrete transform is a mathematical idea, we must have some way to represent an image which makes it plausible to operate on. Thus, we utilize matrices. As mentioned before, an image is a set of pixels collected in rows and columns, so this lends itself well to a matrix. Each pixel value becomes an element of the matrix, and therefore can be manipulated quite easily. In fact, the results of many of the following steps are typically just new matrices composed of transformed pixel values.

What follows will be a breakdown of the steps into further detail to help explain the JPEG compression method. Once one is familiar with the compression method, it isn’t too difficult to envision the decompression method.

VII. A More Detailed Look

A. Reducing the Color Space

Images typically use the RGB color space to represent the colors for a given pixel. RGB is an acronym for “red green blue”, which form a basis for all the colors contained within an image. A 24-bit color image, will store a color value for a pixel in 24-bits. The first 8 bits of the pixel will represent a value for the red contained in the pixel, the next 8 bits for the blue, and the last 8 bits for the green. Blending various intensities of red, green, and blue will yield different colors. For instance, if you have 255-0-0, then that pixel would be pure red. The value, 255, is the maximum value that can be represented in 8 bits. Likewise, 0-255-0 would give us pure blue, and so on and so forth.

However, before even working with an image and the JPEG compression method, it is often useful to transform the color space from RGB to YCbCr. YCbCr is a color space which also utilizes three components, but rather than using three color components, it utilizes one luma component called Y and two chroma components called Cb and Cr. In other terms, it uses one component to represent the intensity or brightness of a pixel, and then two components to represent the color value. The luma component (1) is calculated as a combination of red, green, and blue values. The two chroma components are calculated as blue-difference (2) and red-difference (3), which are also combinations of red, green, and blue values.

$$Y = 0.2989 \times R + 0.5866 \times G + 0.1145 \times B \quad (1)$$
$$Cb = -0.1688 \times R - 0.3312 \times G + 0.5000 \times B \quad (2)$$
$$Cr = 0.5000 \times R - 0.4184 \times G - 0.0816 \times B \quad (3)$$

The reason for this color space change is biological. The construction of the human eye leads us to be more sensitive and responsive to changes in brightness or intensity rather than changes in color. Therefore, it is useful to store the intensity as one of the components used as a descriptor for pixel information.

B. Splitting Up the Image

To best utilize the JPEG method, it is necessary to break any image up into smaller chunks. The generally accepted size is 8 pixels by 8 pixels. Both smaller chunk sizes and larger sizes cause the discrete cosine transform to become less effective and either too much or too little information is lost. This will become clearer as the transform is explained.

If the image cannot be evenly split into 8 by 8 pieces, then the last column and last row of pixels will be duplicated until the desired dimensions are obtained. At that point, each chunk will be subjecting to the remaining steps in the algorithm.

C. Discrete Cosine Transform

After the image is split up, we are working with some number of smaller 8 by 8 matrices of pixel information. Each matrix is used in conjunction with the two-dimensional discrete cosine transform to generate a new matrix of coefficients. These coefficients are the values which will be used to describe the image.

Basically, the discrete cosine transform will give a matrix with low frequency information stored in the coefficients which are closer to the upper left corner of the matrix and high frequency information stored towards the bottom right corner. Frequency, in this context, is related to the occurrence of different pixel values. Values that are more unique are considered lower frequency than those which occur often. We will look more closely at this resulting coefficient matrix in section VIII.

Each coefficient is calculated as a sum of all of the values in the matrix with influence affected by the position of the value and will be a real number. The discrete cosine transform, in two dimensions, is defined as

$$G_{i,j} = \sqrt{\frac{2}{m}}\sqrt{\frac{2}{n}} C_i C_j \times \sum_{x=0}^{n-1}\sum_{y=0}^{m-1} p_{x,y} \cos\left[\frac{(2y+1)j\pi}{2m}\right]\cos\left[\frac{(2x+1)i\pi}{2n}\right] \quad (4)$$

This transform also has an inverse, defined as

$$p_{x,y} = \sqrt{\frac{2}{m}}\sqrt{\frac{2}{n}} \times \sum_{i=0}^{n-1}\sum_{j=0}^{m-1} C_i C_j G_{i,j} \cos\left[\frac{(2y+1)j\pi}{2m}\right]\cos\left[\frac{(2x+1)i\pi}{2n}\right] \quad (5)$$

In each case, C_f is defined by

$$C_f = \begin{cases} \frac{1}{\sqrt{2}}, f = 0 \\ 1, f > 0 \end{cases}. \quad (6)$$

Also, the *i*, *j* values are positional markers in the coefficients matrix and the *x*, *y* values are positional markers in the original matrix.

It is important to note, at this point, the process is still essentially lossless. The only information lost after the initial matrix transform is due to computer precision issues. If we had infinite precision, then the image information could be fully recovered using the inverse transform.

However, after generating the coefficient matrix, a reduction in the magnitude of the coefficients is performed. Typically, the goal is to translate the original values of the coefficient matrix into a new set of values. Low frequency information should still retain significant value relative to the other coefficients, and the higher frequency information should shrink in value or be sent to zero. This reduction occurs in the quantization step.

D. Quantization

The JPEG method wouldn't be much of a compression method if it stopped at this point. Performing the transform on any of the 8 by 8 image chunks still returns an 8 by 8 matrix of coefficients. However, ideally, that coefficient matrix will have significant values towards the upper left and values approaching zero towards the bottom right. Again, this will become clearer in examples. Basically, the bottom right corner represents high frequency (redundant) information, which will be given smaller coefficients (or a weaker weight) when reconstructing the image information.

In fact, since this information is less important, it is often useful to discard it completely. This is done in the quantization step, where the magnitude of all of the values is reduced in some manner. The method of reduction will determine how much compression can be obtained. It can be as simple as rounding each coefficient value down. In general, the more "aggressive" the quantization method, the more likely it is that many of the values will approach zero. This is demonstrated in the JPEG method.

In the case of the JPEG compression method the quantization matrix contains a range of values, typically smaller in size towards the upper left and larger in size towards the bottom right. To perform quantization, each value in the coefficient matrix is divided by its corresponding value in the quantization matrix, i.e. the value in row 1, column 1 of the coefficient matrix would be divided by the value in row 1, column 1 of the quantization matrix, and would become the value of row 1, column 1 of our resultant matrix, and so on for each value. Using this method of quantization, the weaker coefficients in our coefficient matrix are often reduced to 0. Impressively, even with a resultant matrix containing a few significant values and a large section of zeroes, a close approximation of the original information can still be obtained. This is the lossy step of the JPEG algorithm, and the also allows for compression when saving

the information. The matrix used for quantization can be any matrix. Its effectiveness is determined by the values it contains. In other words, different matrices will be used depending on the desired quality reduction in exchange for smaller file sizes.

At this point, the information is still an 8 by 8 matrix, which is made up of 64 values, just like the original image. However, if the image has compressed, then many of these values will be 0. It turns out that collecting the values in a zigzag pattern can form an array where many of those 0 values will end up next to each other. Thus, the array can be further compressed with run-length encoding.

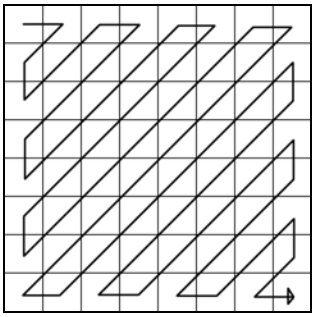


Figure 1. The zigzag pattern makes it possible to collect long runs of zeros which are useful when utilizing run-length encoding.

After the information is run-length encoded it is often further compressed even further using Huffman or arithmetic encoding. This information is then saved as part of the compressed image. Though this writing is not on Huffman encoding nor arithmetic encoding, a brief explanation of each is provided.

Huffman encoding represents pieces of the data by assigning symbols or code words to those pieces. The code words are variable length, and the more frequent the piece of data appears the shorter its code word will be. Therefore, the compressed representation of the data is smaller since it is composed of primarily of the smaller code words.

Arithmetic coding is also a form of variable-length entropy encoding. Rather than using code words as Huffman encoding does, arithmetic coding represents different symbols by taking the interval [0, 1) and splitting it up into sub intervals. Each sub interval represents a symbol. These intervals are used to calculate a single fraction between 0 and 1 which can be used to represent the original data. It is decoded by reversing the calculation steps.

This entire process is repeated for all of the chunks which composed the original image. Ideally, each chunk is smaller than its original 8 by 8, 64 value piece of information, yielding a smaller overall file size. Examples in the following section will reinforce the validity of this process.

VIII. Discrete Cosine Transform Examples

The discrete cosine transform (DCT for short) is applied in two dimensions when working with images. In fact, the two-dimensional transform is performed by doing a one-dimensional transform to each row of the data block, and then to each column of the result. First the one-dimensional transform will be demonstrated.

The DCT, in one dimension, is given by

$$G_f = \sqrt{\frac{2}{n}} C_f \sum_{t=0}^{n-1} p_t \cos\left[\frac{(2t+1) f\pi}{2n}\right] \text{ for } f = 0, 1, \ldots n\text{-}1 \qquad (7)$$

and C_f is still defined as in (6).

The input is a set of data p of size n. The result will be another set of data G of size n. The values of G are the DCT coefficients. The G_0 coefficient is typically called the DC coefficient and the rest are AC coefficients. This terminology comes from electrical engineering. The coefficients are real numbers and can be positive or negative. The f and t are used to reference position in either the output data set or the input data set respectively.

Conveniently, there also exists an inverse one-dimension DCT given by

$$p_t = \sqrt{\frac{2}{n}} \sum_{j=0}^{n-1} C_j G_j \cos\left[\frac{(2t+1) j\pi}{2n}\right]. \qquad (8)$$

This equation can be used to reconstruct the original values using the set of transform coefficients produced by the DCT.

A. One Dimensional Examples

For a simple example, let p = {5, 6, 5, 4, 5, 6, 6, 3}. Applying the one-dimensional DCT (1), the resulting set of coefficients is {14.1421, 0.6055, -0.2706, 1.5996, -2.1213, 0.0423, -0.6533, 0.2697}. For purposes of this writing, these coefficients have been rounded down to four decimal places. In general, using the inverse one-dimensional DCT (IDCT for short) (3) with this set would give back the original values. There may be slight errors due to both rounding, as well as any errors caused by machine precision.

The main advantage of the JPEG method is that once the transform coefficients are generated, they are quantized in order to compress the data. For the first attempt at quantization, round each coefficient to the nearest integer giving us the set {14, 1, 0, 2, -2, 0, -1, 0}. Using these coefficients and the IDCT, the reconstructed values are {5.3732, 6.3394, 4.4919, 3.9760, 4.8920, 5.8979, 5.8981, 2.7294}. Quantizing the coefficients even further we can use {14, 0, 0, 2, -2, 0, 0, 0} and reconstruct values of {5.0741, 5.4618, 4.6761, 3.6871, 4.7982, 6.6376, 5.8519, 3.4112}. The biggest difference between reconstructed and original values is the 0.6376 (between 6.6376 and 6). This is quite impressive since the reconstructed values only required three non-zero coefficients.

It should be noted that the transform and quantization yield better results for correlated data. The less correlated the data is, the worse the reconstruction will be after quantization. This coincides with the types of images which work well with JPEG compression versus those that do not, i.e. in continuous tone images neighboring pixels tend to have similar values, where as in a line art image, since there are often sharp edges of color, the pixel values may be quite unrelated.

Another quick one-dimensional example with uncorrelated data will help demonstrate. Let p = {81, 200, 0, 2, 65, 120, 40, 1}. After inputting p into the DCT, the resulting coefficients are {179.9587, 66.2694, 29.8901, 93.9991, -74.5998, -99.1367, -52.5626, -55.6354}. Quantizing these by rounding to the nearest integer gives the set {180, 66, 30, 94, -75, -94, -53, -56}. Using those coefficients, the reconstructed values are {80.7108, 200.3013, -0.2804, 2.1151, 64.6969, 120.1465, 40.4570, 0.9695}. These are still fairly close to the original values. However, no coefficients have actually been eliminated. Quantizing again, rounding to the nearest multiple of ten and keeping only coefficients which are greater than or equal to 50 gives us the set {180, 70, 0, 90, -80, -90, -50, -60}. Now, a coefficient has been sent to 0. However, reconstructing from these values yields the following values {66.6785, 196.1457, 10.4134, 18.7574, 71.0875, 127.2404, 33.8961, -15.1020}. Indeed, this uncorrelated data does not lend itself well to the transformation and quantization of the JPEG method.

B. Two Dimensional Examples

The following will illustrate a couple of two dimensional examples. There may be a small margin of error due to the use of machine precision. Each example will be given as a series of tables representing the original image values, the initial transform coefficients, the quantized coefficients, and the reconstructed image. After that, there will be a table that highlights the differences from original image to reconstructed image. The first example (Figure 2) is utilizing correlated data that is an approximation of a smooth gradient of color. This example would be analogous to a continuous tone image where there is a lot of correlation between the pixel values.

The second example (Figure 3) is an image with sharply defined regions of solid color, perhaps similar to a piece of graphic art or line art. This is not as well suited to the JPEG compression method because of the more drastic color variations at the boundaries of color regions, as the example will demonstrate.

215	201	177	145	111	79	55	41
215	201	177	145	111	79	55	41
215	201	177	145	111	79	55	41
215	201	177	145	111	79	55	41
215	201	177	145	111	79	55	41
215	201	177	145	111	79	55	41
215	201	177	145	111	79	55	41
215	201	177	145	111	79	55	41

(a)

0	4994	0	1.6	0	1.2	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	0	0	0
0	0	0	0	0	0	0	0

(b)

0	45	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	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(c)

213	200	176	145	110	79	55	42
213	200	176	145	110	79	55	42
213	200	176	145	110	79	55	42
213	200	176	145	110	79	55	42
213	200	176	145	110	79	55	42
213	200	176	145	110	79	55	42
213	200	176	145	110	79	55	42
213	200	176	145	110	79	55	42

(d)

2	1	1	0	1	0	0	−1
2	1	1	0	1	0	0	−1
2	1	1	0	1	0	0	−1
2	1	1	0	1	0	0	−1
2	1	1	0	1	0	0	−1
2	1	1	0	1	0	0	−1
2	1	1	0	1	0	0	−1
2	1	1	0	1	0	0	−1

(e)

Figure 2. The original image (a) is analogous to a continuous tone image with a smooth gradient of color. The DCT coefficient matrix is computed (b) and then those results are quantized (c). The quantized coefficient matrix can be used to reconstruct the image (d). The reconstruction differs very little from the original image (e).

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 255 & 255 & 255 & 255 & 0 & 0 & 0 \\ 0 & 255 & 0 & 0 & 0 & 255 & 0 & 0 \\ 0 & 255 & 0 & 0 & 0 & 0 & 255 & 0 \\ 0 & 255 & 0 & 0 & 0 & 0 & 255 & 0 \\ 0 & 255 & 0 & 0 & 0 & 255 & 0 & 0 \\ 0 & 255 & 255 & 255 & 255 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(a)

$$\begin{bmatrix} -514 & 149.9 & -97.5 & -35.1 & -255 & -176.8 & -235.5 & -100. \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -97.5 & 54.2 & -180.3 & -95.7 & 235.5 & 19 & 180.3 & 81.1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -255 & -149.9 & 235.5 & 35.1 & 0 & 176.8 & -97.5 & 100.1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -235.5 & -130.8 & 180.3 & 231 & -97.5 & -45.9 & 180.3 & -196 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(b)

$$\begin{bmatrix} -32 & 13 & -9 & -2 & -10 & -4 & -4 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -6 & 4 & -11 & -3 & 5 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -14 & -6 & 6 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -4 & -2 & 2 & 2 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(c)

$$\begin{bmatrix} 1 & 3 & 20 & -2 & 17 & 0 & -12 & 15 \\ -1 & 267 & 225 & 226 & 222 & 40 & 20 & -25 \\ 26 & 211 & 7 & 28 & 43 & 179 & 51 & 3 \\ 4 & 226 & 34 & -21 & -14 & 25 & 202 & 19 \\ 4 & 226 & 34 & -21 & -14 & 25 & 202 & 19 \\ 26 & 211 & 7 & 28 & 43 & 179 & 51 & 3 \\ -1 & 267 & 225 & 226 & 222 & 40 & 20 & -25 \\ 1 & 3 & 20 & -2 & 17 & 0 & -12 & 15 \end{bmatrix}$$

(d)

$$\begin{bmatrix} -1 & -3 & -20 & 2 & -17 & 0 & 12 & -15 \\ 1 & -12 & 30 & 29 & 33 & -40 & -20 & 25 \\ -26 & 44 & -7 & -28 & -43 & 76 & -51 & -3 \\ -4 & 29 & -34 & 21 & 14 & -25 & 53 & -19 \\ -4 & 29 & -34 & 21 & 14 & -25 & 53 & -19 \\ -26 & 44 & -7 & -28 & -43 & 76 & -51 & -3 \\ 1 & -12 & 30 & 29 & 33 & -40 & -20 & 25 \\ -1 & -3 & -20 & 2 & -17 & 0 & 12 & -15 \end{bmatrix}$$

(e)

Figure 3. The original image (a) is analogous to a sharp, graphic image with few colors and sharp color borders. These types of images do not typically work as well with JPEG compression. The DCT coefficient matrix is computed (b) and then those results are quantized (c). The quantized coefficient matrix can be used to reconstruct the image (d). The reconstruction of this image (e) is much less accurate than that of Figure 2.

It is quite telling to compare the two difference matrices. There is very little difference in value variation in the correlated example. The smooth gradient would certainly still be visible. However, in the uncorrelated example, the sharpness is lost. The shape may still be visible, but the image would appear quite blurry. The differences in the correlated image were at most 2 (not quite 1% of its original value is missing). In the uncorrelated image, the maximum difference is 76 (a much larger 30% of its original value is lost). Clearly, the more correlated the image information, the better the results.

IX. A Brief Explanation of the DCT

In this section, I will attempt to provide a little insight into the discrete cosine transform. A full explanation is outside of the scope of this paper. However, I will attempt to highlight the interesting points of the transform, as well as provide a couple of alternative ways of viewing the transform.

The major advantage of the DCT is that it packs a lot of significant information about the image into the first few coefficients in the resulting matrix. Therefore, even if many of the other coefficients are lost during quantization, the most important information about the image is retained and a close approximation of the original can be reconstructed during decompression.

To get a better feeling for this, think of the DCT as working with a set of basis functions to construct the original information. The basis functions are the building blocks for reconstructing the information. Each basis function takes a sampling of the original values and each coefficient will serve as a scalar multiplier to each basis function. These pairs are combined as a linear combination to reconstruct the original information. The first basis function takes an even sampling of all the values and each subsequent function gives more influence to those values that are more frequently occurring. Thus the first few coefficients tend to be the most important since they contain information about both low and high frequency information. Since that high frequency information is already given representation in the earlier coefficient and function pairings, the later functions that focus entirely on high frequency information can more or less be forgotten. This is even more apparent when examining the coefficient matrices from the two dimensional examples [10].

When dealing with two-dimensional DCT, there are *m* by *n* basis functions. For example, with JPEG compression there are 64 basis functions, and thus the JPEG method attempts to

find coefficient-basis function pairs to recreate the original image approximately. It helps to imagine them arranged in an 8 by 8 grid. The right-most functions in the grid are very horizontally high frequency focused and the bottom-most functions in the grid are most concerned with vertical frequency. For example, the upper-left corner is an even sampling and the bottom-right corner is concerned with values that appear both vertically and horizontally at high frequencies. The more frequently a value occurs, the less information is required to represent it. Looking at the coefficient matrix in Figure x, the significant coefficients are dealing with horizontal frequency. Since the original image is changing from left to right, the basis functions gathering information about horizontal frequency are more important and therefore given larger coefficients. More specifically, since the original image was a gradual change from high value to low value (i.e. low frequency) only one low frequency basis function was needed to make a fairly accurate reconstruction of the original image.

One other way to help visualize the DCT is as an axis rotation. Keeping with the idea of vector spaces and coordinates from above, an 8 by 8 image is represented by 64 different coordinates. Unfortunately, a space with 64 axes is hard to imagine. Scaling back, think of a 2 axes graph with x and y coordinates. On this graph, all of the values in our image would be represented by x, y coordinate pairs. Imagine rotating the axis-frame so that the x-axis runs through the middle of all those correlated values. Most of their y coordinates are now close to 0 and the only significant information is the x coordinate. Thus, we can get a close approximation of the values with only one of our two original coordinates. Extending this idea back into the 64-dimensional space of our 8 by 8 blocks, the DCT attempts to eliminate the need for most of the 64 coordinates and represent the values with only a few.

X. Future Interests

The JPEG compression method has been around for over 15 years now, so one can't help but wonder if something new and improved has been developed. In fact, there is another method known as JPEG-2000, which yields still higher compression ratios with little loss in quality. The method is similar to the JPEG method, but the main transformation is a wavelet transformation instead of a discrete cosine transformation [5]. I would be very interested to see why this yields better results, and further still if this is true, why it hasn't been accepted and implemented as widely as the current JPEG compression method. I am also interested in experimenting and writing my own image compression implementation. Additionally, I am interested in learning more on the topic of Fourier analysis to better understand the workings of the discrete cosine transform.

XI. Conclusion

At the end of this study, it was obvious why this method of compression is so prevalent. JPEG yields significant compression while still maintaining much of the quality. The method itself is fairly straight forward as a process, with much of the complexity being introduced when attempting to understand the inner workings of the discrete cosine transform. It was impressive and surprising to me that so much information could be discarded, and yet a close approximation of that information could still be recovered from the small amount retained.

Acknowledgment

I would like to thank Professor Kristine Peters and Professor David Scott for all of their assistance and advice throughout the course of this study. I would also like to thank my fellow group members, Curtis Schmitt, Colin Harter, and Arthur Rumpf IV, for their support and contributions to our group research. Additionally, I would like to thank the other members of the Math and Computer Science department for their thoughts and advice.

References

[1] Blelloch, Guy E. Introduction to Data Compression. Carnegie Mellon University, 2001.
[2] Cabeen, K. and Gent, P. Image Compression and the Discrete Cosine Transform. College of the Redwoods.
[3] Hankerson, Darrel R. Introduction to Information Theory and Data Compression. Boca Raton, Fla: Chapman & Hall/CRC P, 2003.
[4] Hoggar, S. G. Mathematics of Digital Images Creation, Compression, Restoration, Recognition. New York: Cambridge UP, 2006.
[5] "JPEG home page". The JPEG committee home page. 3 Feb. 2009 Available: <http://www.jpeg.org/jpeg/index.html>
[6] Khayam, Syed A. The Discrete Cosine Transform (DCT): Theory and Application. Michigan State University, 2003.
[7] Null, Linda. Essentials of Computer Organization and Architecture. Sudbury, MA: Jones and Bartlett, 2006.
[8] Salomon, David. Data Compression: The Complete Reference. New York: Springer, 2006.
[9] Shannon, C. E. "A Mathematical Theory of Communication." The Bell Systems Technical Journal (1948).
[10] Symes, Peter. Digital Video Compression. New York: McGraw-Hill/TAB Electronics, 2003.
[11] Wolfgang, Ray. "JPEG Tutorial". The Society of Imaging Science and Technology. 3 Feb. 2009. Available: <http://www.imaging.org/resources/web_tutorials/jpegtutorial/>

Daniel S. Syens was born near Seoul, South Korea on January 9, 1987. He was adopted and brought home to Friesland, Wisconsin on July 10, 1987. He graduated from Cambria-Friesland High School in Cambria, Wisconsin in 2005. Daniel is currently attending Ripon College in Ripon, Wisconsin and will be receiving a Bachelor of Arts degree with a major in computer science and a minor in mathematics.

He has spent the past few summers working for Advanced Energy Control assisting with CAD and various programming tasks. During the school year, he works for the Ripon College Math and Computer Science Department as a department assistant. In addition, he also works for Ripon College Information Technology Services as a student assistant and for Ripon College Student Support Services as a tutor. After graduate, he plans to assume a full-time position at Advanced Energy Control.

Mr. Syens is a member of the Ripon chapter of Phi Beta Kappa. In his spare time, he enjoys reading, music, movies, staying active, and spending time with close friends and family.

Summation
May 2009, pp. 32-35
http://ripon.edu/macs/summation

A Summary of the Alexander Polynomial and Related Knot Invariants

Eli Annis
Department of Mathematics and Computer Science – Ripon College

***Abstract*—James Alexander's method of calculating the Alexander polynomial and the method for calculating the Alexander-Conway polynomial are presented. The relationship between the Alexander polynomial and the Jones and HOMFLY polynomials is also discussed.**

***Key Words*—Alexander polynomial, HOMFLY polynomial, Jones polynomial, knot theory.**

I. Introduction

KNOT polynomials are used to determine if two knots are equivalent. Knots are said to be equivalent, if and only if the knots can be transformed into each other by a series of Reidemeister moves. However, knot polynomials simplify the process of determining knot equivalence by eliminating the need the determine an actual series of Reidemeister moves that will transform the knots. If two knots are represented by polynomials that aren't equivalent, then the knots aren't equivalent [1].

The Alexander polynomial was developed by James W. Alexander II and published in his paper "Topological invariants of knots and links" in 1928 [2]. Not only was it the first polynomial invariant of knots, but from its publication in 1928 until the publication of the Jones polynomial in 1985 the Alexander polynomial was also the only known polynomial knot invariant [3].

This paper will first explain a method of calculating the Alexander polynomial using matrices as outlined in Alexander's original 1928 paper. It will then describe an alternative method of calculation, which makes use of a skein relation. Last, the relationship between the Alexander polynomial and the Jones and HOMFLY polynomials will be explained.

II. James Alexander's Method of Calculation

The method developed by James Alexander for calculating the Alexander polynomial of a knot consists of four steps: labeling the knot diagram, creating the equations of the diagram, expressing the set of equations as a matrix, and deriving a polynomial from the matrix.

A. Labeling the Knot Diagram

In order to label a knot diagram, it is necessary to choose an orientation for the knot. The chosen orientation is not used in the calculations. However, we indicate the lower arcs at each crossing using dots on the left hand side of the arc. So the lower arc at a crossing is parallel to the two dots while the upper arc in a crossing will pass between the two dots. Consequently, the orientation determines the side of the arc on which the dots lie.

A knot diagram partitions the plain into two or more regions. We label these regions $r_0,r_1,...,r_{n-1}$ where n is the number of regions. Similarly, we label the crossings c_1, c_2,..., c_m where m is the number of crossings in the knot diagram [2]. A labeled diagram of a trefoil knot is shown in figure 1.

B. Creating the Equations of the Diagram

After a diagram is labeled, we are able to create a set of equations describing the crossings. The equations of the crossings will all have the form

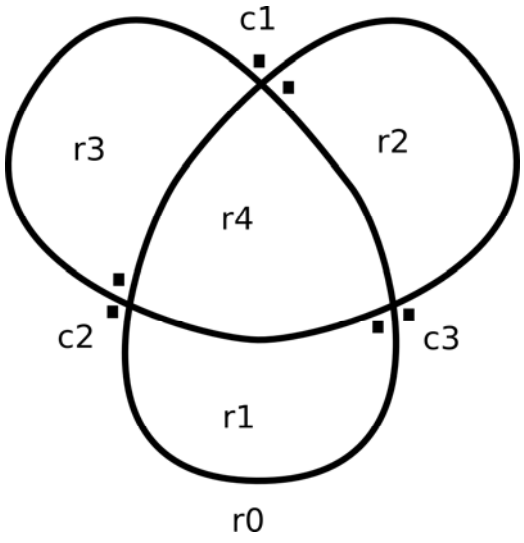


Figure 1. Labeled knot diagram of a left-handed trefoil knot.

Work presented to the college March 10, 2009.

E. Annis will be pursing an MS in mathematics. at the University of Wisconsin, Milwaukee. phone: 715-755-2410; e-mail: eli_annis@ hotmail.com .

$$c_a(x) = xr_b - xr_c + r_d - r_e, \quad (1)$$

where a is the index of the crossing and b, c, d, and e are the index of the regions around the crossing c_a in counter-clockwise order, and r_b and r_c are the regions containing the dots which distinguish between the upper and lower arcs of the crossing [2]. These equations are called the equations of the diagram. For the trefoil knot in figure 1, the equations of the diagram are

$$c_1(x) = xr_2 - xr_0 + r_3 - r_4, \quad (2)$$

$$c_2(x) = xr_3 - xr_0 + r_1 - r_4, \quad (3)$$

and

$$c_3(x) = xr_1 - xr_0 + r_2 - r_4. \quad (4)$$

C. Expressing the Equations as a Matrix

The equations of the diagram are a set of linear equations. Therefore, we can represent these equations using a matrix, M, where the ij-th element of the matrix is the variable or constant in the term of equation c_i with the coefficient r_j If there is no term with a coefficient of r_j in equation c_i, then the corresponding element of the matrix is 0 [2]. Consequently, each row of M corresponds to a crossing and each column corresponds to a region. So M is an m x n matrix. Using this method,

$$M = \begin{bmatrix} -x & 0 & x & 1 & -1 \\ -x & 1 & 0 & x & -1 \\ -x & x & 1 & 0 & -1 \end{bmatrix} \quad (5)$$

is the matrix formed by (2), (3), and (4).

D. Deriving the Polynomial from the Matrix

Only square matrices have determinants and M is not a square matrix. Euler's theorem of polyhedra states that for any polyhedral the faces, edges, and vertices are related by the equation

$$V + F - E = 2, \quad (6)$$

where, V is the number of vertices of the polyhedra, F is the number of faces, and E is the number of edges [4]. Using (6) on the surface bound by the arcs of a knot, it is possible to show that

$$n = m + 2. \quad (7)$$

Since M is an n by m matrix, if we eliminate two columns in M, then the resulting matrix, M_1, will be an m by m matrix and we can find its determinant [2]. Removing the last two columns of (5) will result in the matrix

$$M_1 = \begin{bmatrix} -x & 0 & x \\ -x & 1 & 0 \\ -x & x & 1 \end{bmatrix}. \quad (8)$$

In his paper, James Alexander showed that we can remove any two columns from M and the determinant of M_1 will only differ by a factor of $\pm x^p$ where p is some integer [2]. Therefore, we normalize the determinant of M_1 by multiplying by some power of x, and -1 if necessary, so that the resulting polynomial has a positive constant term. This polynomial is the Alexander polynomial and is typically denoted by $\Delta(x)$. Using (5),

$$\Delta(x) = 1 - x + x^2. \quad (9)$$

is the Alexander polynomial for the knot in figure 1.

III. Alexander-Conway Polynomial

The method developed by James Alexander works reasonably well for a single knot with only a few crossings. However, the method is inefficient for calculating the Alexander polynomial for multiple knots or knots with a large number of crossings. Also, the method would be somewhat difficult to implement in a computer program.

In the 1960s John Conway developed a recursive method of calculating the Alexander polynomial, which resolves some of the shortcomings of Alexander's method [5]. The Alexander polynomial resulting from this method is often known as the Alexander-Conway polynomial.

A. Skein Relation

There are three possibilities for a crossing in a knot diagram, a right-handed crossing, a left-handed crossing, and no crossing. In a left-handed crossing, the lower curve has an orientation going from left to right when the orientation of both curves at the crossing leads away from the viewer. A right-handed crossing has the orientation of the lower curve going from right to left and in the final type of crossing, the two curves involved do not cross at all.

A skein relation is a relationship between the polynomial invariants of three knots that differ at only a single crossing, where the crossing is right-handed in one knot, left-handed in the second knot, and there is no crossing in the third knot [6]. A skein relation will have the form

$$\nabla_{D+}(x) - \nabla_{D-}(x) = x\nabla_{D0}(x), \quad (10)$$

where the knot with the right-handed crossing is denoted D_+, the knot with the left-handed crossing is denoted D_-, and the third crossing is denoted D0 [5]. An example using the trefoil knot from figure 1 can be found in figure 2.

B. Alexander-Conway Polynomial

The Alexander-Conway polynomial for a knot, D, can be calculated using three rules. The first rule is that any knot

equivalent to the unknot has a polynomial invariant of 1. The second rule is that any knot equivalent to two unknots that are not linked has a polynomial invariant of 0. The last rule is the skein relation

$$\nabla_{D+}(x) - \nabla_{D-}(x) = (\sqrt{x} - 1/\sqrt{x})\nabla_{D0}(x). \quad (11)$$

To calculate the polynomial using these rules, we first choose a crossing in our knot, D, and solve the skein relation in rule 2 for that particular crossing. This is known as a skein operation [5]. Using the trefoil knot in figure 1 as an example, the a skein operation on any crossing will result in the equation

$$\nabla_{D-}(x) = \nabla_{D+}(x) - (\sqrt{x} - 1/\sqrt{x})\nabla_{D0}(x) \quad (12)$$

as all of the crossings in the knot in figure 1 are left-handed. After a skein operation is performed for the chosen crossing, we choose a crossing in each of the alternative versions of D, in this case D+ and D0, and perform a skein operation both of these knots. This process is repeated for each new set of alternative knots until each knot has been reduced to a knot equivalent to the unknot or a knot equivalent to two unknots, which are not linked. Then by rules 1 and 2, the polynomial of each of these knots is either 1 or 0. It is then possible to repeatedly back-substitute until we reach and equation relating a crossing in D to a polynomial [7]. This polynomial is the Alexander-Conway polynomial for D and is denoted ∇(x).

In the example of the trefoil knot, we performed a skein operation for ∇D–, so a crossing is chosen from D+ and D0, which are shown figure 2, and a skein operation is performed on both of these knots. The skein relation solved for a crossing in ∇D0 would be

$$\nabla_{D0-}(x) = \nabla_{D0+}(x) - (\sqrt{x} - 1/\sqrt{x})\nabla_{D00}(x) \quad (13)$$

as D contained only left-handed crossings so all of the crossings in D0 will be left-handed as well. The knot D+ is equivalent to the unknot so according to rule 1,

$$\nabla_{D+}(x) = 1. \quad (14)$$

So we can substitute 1 for ∇D+ in (12). Similarly, D0+ is equivalent to two unknots that are not linked and D00 is equivalent to an unknot. Consequently,

$$\nabla_{D0+}(x) = 0 \quad (15)$$

and

$$\nabla_{D00}(x) = 1. \quad (16)$$

We then enter (14) and (15) into (13), which results in the polynomial

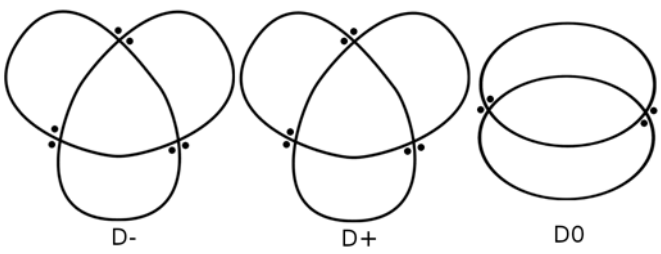


Figure 2. Example of knots related by a skein relation. D- is the left-handed trefoil knot while D+ and D0 are the knots resulting from a skein operation on D-.

$$\nabla_{D0-}(x) = 1. \quad (17)$$

Finally, we use (17) in (12) resulting in (12)

$$\nabla(x) = -1 + x + x^{-1}, \quad (18)$$

which is the Alexander-Conway polynomial for the knot in figure 1.

IV. Related Knot Invariants

Although John Conway's version of the Alexander polynomial is easier to calculate the original, the two polynomials are the same. Consequently, they have the same limitations. Neither polynomial is capable of distinguishing between a knot and its mirror image such as the two types of trefoil knots and both polynomials are useful on knots of only eight or fewer crossings [1],[2]. The weaknesses of the Alexander polynomial were eventually overcome by the development of the Jones and HOMFLY polynomials in the 1980's.

The Jones polynomial was developed by Vaughn Jones and published in his article “A polynomial invariant for knots via Von Neumann algebras” in 1985 [8]. The Jones polynomial addresses both of the weaknesses of the Alexander polynomial as it is capable of distinguishing between right and left-handed knots and distinguishes between knots of 10 or fewer crossings [1]. However, like the Alexander-Conway polynomial, we can calculate the Jones polynomial recursively using only two rules. Rules 1 and 2 for calculating the Jones polynomial are the same as rules 1 and 2 for the Alexander-Conway polynomial. But, rule 3 is modified to

$$x^{-1}\nabla_{D+}(x) - x\nabla_{D-}(x) = (\sqrt{x} - 1/\sqrt{x})\nabla_{D0}(x). \quad (19)$$

The similarity in the forms of the skein relations for Alexander-Conway and Jones polynomials was quickly noticed and only months after the Jones polynomial was published, Raymond Lickorish, Ken Millett, Jim Hose, Peter Freyd, David Yetter, and Adrian Ocneanu collectively published a paper titled “A new polynomial invariant of knots and links”, which introduced a generalized form of the Alexander and Jones Polynomials [9]. Jozef Przytycki and Pawel Traczyk also developed this same polynomial, but were not part of the initial publication [1]. The polynomial these

mathematicians developed is known as the HOMFLY polynomial or HOMEFLYPT polynomial after the initials of the authors of it's discoverers. This polynomial knot invariant uses the skein relation

$$v^{-1}\nabla_{D+} - v\nabla_{D-} = z\nabla_{D0}. \quad (20)$$

Acknowledgment

I would like to thank Professor Karl Beres for his help with this paper. I would also like to thank the other members of my group, Peter Babcock and Greg George. Finally, I want to thank the Mathematics and Computer Science faculty at Ripon College for all their help during my undergraduate studies.

REFERENCES

[1] E. W. Weisstein, "Alexander polynomial," from *MathWorld*. Available: http://mathworld.wolfram.com/AlexanderPolynomial.html

[2] J. Alexander, "Topological invariants of knots and links," *Transactions of the American Mathematical Society*, vol. 3, Apr. 1928, pp. 275–306. Available: http://www.jstor.org/stable/1989123

[3] P. Cromwell, *Knots and Links.* Cambridge, UK: Cambridge University Press, 2004, pp. 236–237

[4] E. W. Weisstein, "Polyhedral Formula," from MathWorld. Available: http://mathworld.wolfram.com/PolyhedralFormula.html

[5] K. Murasugi, *Knot Theory & Its Applications.* Boston: Birkhauser, 1996, pp. 108-109

[6] E. W. Weisstein, "Skein relationship," from *MathWorld*. Available: http://mathworld.wolfram.com/SkeinRelationship.html

[7] R. Messer and P. Straffin, *Topology Now!*, Washingtion, D.C: Mathematical Association of America, 2006, pp. 65-88

[8] L. H. Kauffman, "New invariants in the theory of knots," *The American Mathematical Monthly*, vol. 95, Mar. 1988, pp. 195-242

[9] P. Freyd, et al., "A new polynomial invariant of knots and links," *Bulletin of the American Mathematical Society*, vol. 12, Apr. 1985 Available: http://www.ams.org/bull/1985-12-02/S0273-0979-1985-15361-3/S0273-0979-1985-15361-3.pdf

Eli Annis was born on September 15, 1986 in Osceola, Wisconsin and graduated from Osceola High School in May of 2005. Eli Annis is currently pursuing a bachelor's degree in mathematics at Ripon College in Ripon, Wisconsin.

He has worked as a Departmental Assistant for the Mathematics Department at Ripon College for two years. Upon graduation he will attend graduate school at the University of Wisconsin in Milwaukee.

Mr. Annis has academic interests in topology and astrophysics. His extra-curricular activities include martial arts, video games, and camping.

Summation
May 2009, pp. 36-39
http://ripon.edu/macs/summation

The Reidemeister Moves

Gregory A. George
Department of Mathematics and Computer Science – Ripon College

***Abstract*—Reidemeister Moves are used to deform knots in order to determine knot equivalence. In this paper I will discuss the basis of the Reidemeister moves, knot equivalence, and link equivalence. Using examples I will show that utilizing the Reidemeister moves to prove equality is relatively simple.**

***Key Words*—Knots, Links, Reidemeister Moves, Triangular Moves**

I. Introduction

TOPOLOGY is the study of various geometric objects as they are transformed by continuous deformations [1]. Some of these geometric shapes fall in the category of knots. A knot K is a simple closed curve in $\Re^3$ that can be broken into a finite number of straight line segments $e_1, e_2, \ldots, e_n$ such that the intersection of any segment e_k with the other segments is exactly one endpoint of e_k intersecting an end point of e_{k-1} (or e_n if $k = 1$) and the other endpoint of e_k intersecting an end point of e_{k+1} (or e_1 if $k = n$) [1]. In order to better explain what a knot actually is, it is necessary to understand what is meant by "simple closed curve." In topology this means a connected curve that ends at the same point it begins without intersecting itself [2].

The concept of intersection in the context of knots can be confusing. A knot never intersects itself as a 3-dimensional object. However, knots tend to be modeled by knot diagrams. Knot diagrams are the 2-dimensional representation of the 3-dimensional shape. In these representations there are points where the line segments of the knot cross, in which case the segments are referred to as intersecting. Figure 1 is a knot diagram with seven line segments and seven crossings.

The concept of multiple knots inevitably leads to the idea of knot equivalence. Two knots are said to be equivalent if and only if there is a finite number of triangular moves that changes the first knot into the second knot [1]. This definition begs the question: what are triangular moves?

II. Triangular Moves

Triangular moves are a method of deformation when transforming knots. There are two types of triangular moves: triangular detour and triangular short cut. In order to understand triangular detours, we must consider the triangle

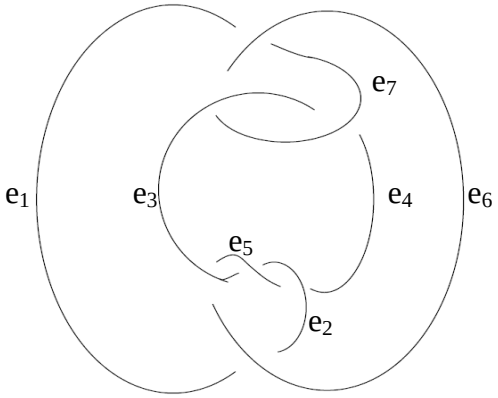


Figure 1. Knot diagram of a knot which contains seven segments and seven crossings.

ABC with the side AC matching one of the line segments of a knot K. A triangular detour involves replacing the edge AC with the two edges AB and BC [1]. This deforms the knot such that the previous segment of the knot K is now the two edges AB and BC. In order to understand triangular shortcuts, we must consider the triangle ABC with the sides AB and BC matching one of the line segments of a knot L. A triangular shortcut involves replacing the two edges AB and BC with the single edge AB [1]. This deforms the knot such that the previous segment of the knot L is now the single edge AB. Performing a triangular detour on knot K will produce knot L, and performing a triangular shortcut on knot L will produce knot K (Figure 2).

Using triangular moves to determine knot equivalence is grossly inefficient. In order to improve efficiency, Kurt Reidemeister examined the use of triangular moves in knot deformations while employed as a professor at the University of Konigsberg in Germany.

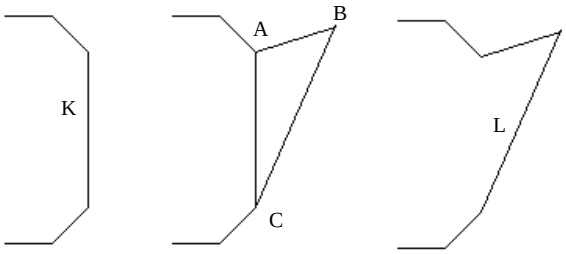


Figure 2. A triangular detour on knot K to form knot L. A triangular cut on knot L to form knot K.

Work presented to the college March 13, 2009. This is the second paper in the *Knot Theory* series. For further information, refer to The Alexander Polynomial and Related Polynomial Knot Invariants written by Eli Annis.

G. A. George graduated from Arundel High School, Gambrills, MD.

III. Reidemeister Moves

In 1927, K. Reidemeister developed what are now known as the Reidemeister moves [3]. The three types of Reidemeister moves simplify the effect of triangular moves in relationship to the whole knot. The three types of Reidemeister moves are the twist (Type I), moving a strand (Type II), and a crossing (Type III).

As the name of the Type I move suggests, the move removes or adds a crossing in a knot which is caused by a twist. The Type II move removes two crossings by moving one segment across or underneath the other. The Type III move moves a segment through an intersection, keeping the relationship between the segments. In this case, the segment that is moved across the intersection remains on top of the other segments while it is moved downwards. See Figure 3.

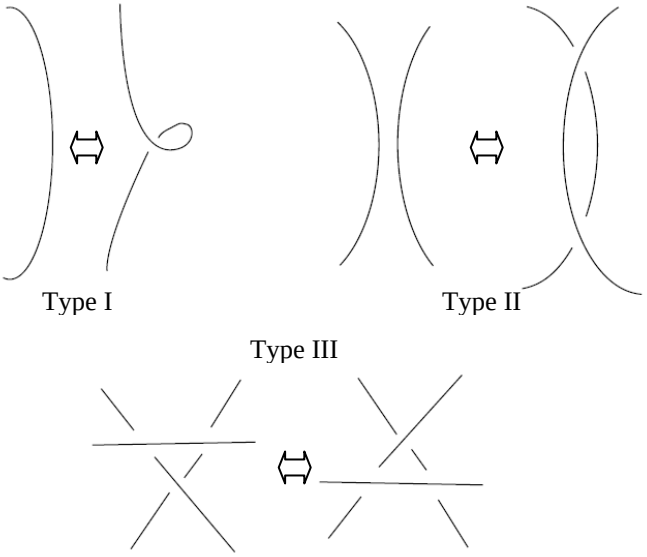


Figure 3. Shows the typical images used to exemplify the Reidemeister moves.

With the Reidemeister moves replacing the triangular moves, we may now change the definition of knot equivalence. Two knots are equivalent if and only if there is a finite number of *Reidemeister moves* that change the first knot into the second knot. The use of the "if and only if" phrasing implies that all of the Reidemeister moves are reversible.

In order to better understand knot equivalence, we will prove that the knot in Figure 1 is equivalent to the trivial knot (Figure 4).

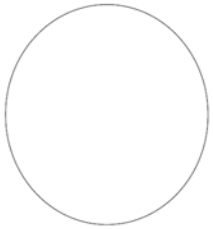


Figure 4. The trivial knot, also known as the unknot.

The first segment of the knot that will be deformed is e_6. As this segment moves from right to left across the knot diagram, several Reidemeister moves will be completed at once. Type II moves will occur with segments e_4, e_3, and e_1. And type III Reidemeister move will occur with all crossings in the knot diagram. Note that Figure 5 shows that the number of crossings and segments in the knot diagram are preserved. In order to show that this knot is equal to the trivial know we will have to eliminate crossings in the knot diagram.

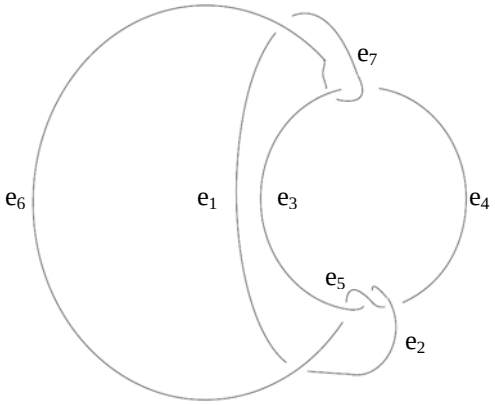


Figure 5. The knot diagram after the first series of Reidemeister moves.

The next line segment that will be deformed is e_1. As this segment moves behind the knot diagram, the type II move will occur with segments e_3 and e_4. Again, type III occurs as e_1 moves behind the crossings in the knot diagram. Note that Figure 6 has eliminated crossings, and that the segments e_1, e_2 and e_7 have become the segment $e_1 + e_2 + e_7$. The original knot is one step closer to becoming the trivial knot.

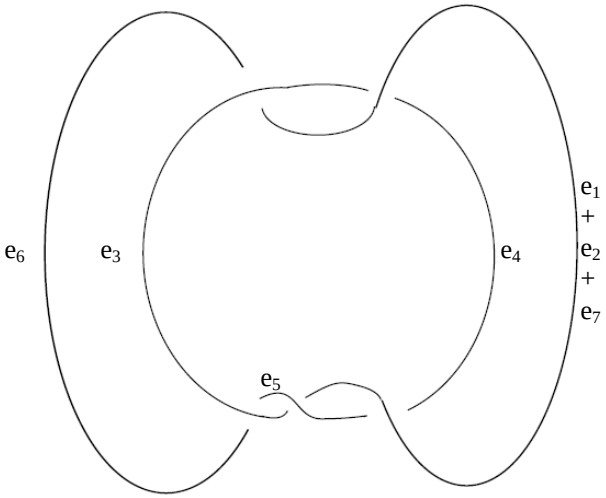


Figure 6. The knot diagram after the second series of Reidemeister moves.

The next step involves one type II move between the segments $e_1 + e_2 + e_7$ and e_4. Note that two crossings have been eliminated, and the segments e_3, e_4, and e_5 combine into the line segment $e_3 + e_4 + e_5$ in Figure 7.

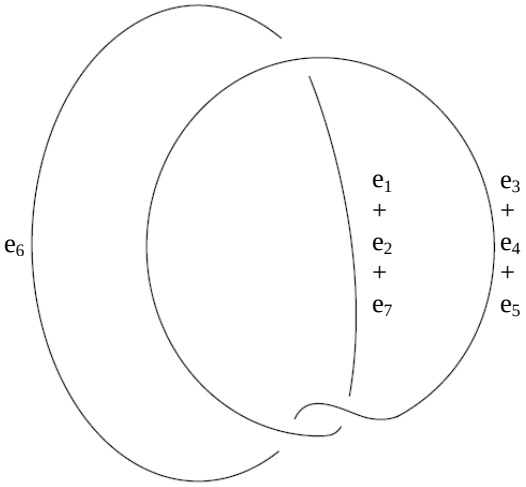


Figure 7. The knot diagram after the third series of Reidemeister moves.

Next, we will manipulate the plane to form a figure eight. Two crossings will be eliminated by this move. The first crossing that will be eliminated is between $e_3 + e_4 + e_5$ and e_6. The second crossing that will be eliminated is between $e_3 + e_4 + e_5$ and $e_1 + e_2 + e_7$. Note that the knot is now all one segment, the segment $e_1 + e_2 + e_3 + e_4 + e_5 + e_6 + e_7$ in Figure 8.

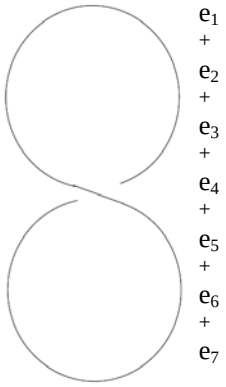


Figure 8. The knot diagram after the fourth series of Reidemeister moves.

Last we will perform a type I transformation, i.e. a twist, on the figure eight. By doing so, we have created the trivial knot seen in Figure 4.

IV. Links

Links are defined as the nonempty union of a finite number of disjoint knots [1]. A case where a link can be produced from one knot is in Figure 4, in which case the trivial knot would be renamed as the trivial link. There are various other famous link combinations to include the Hopf and Borromean Links.

The Hopf Link seen in Figure 9 is produced by interlinking two separate trivial links. The Hopf link has the property of being a simple link, meaning it can not be transformed through the Reidemeister moves to a more simple form.

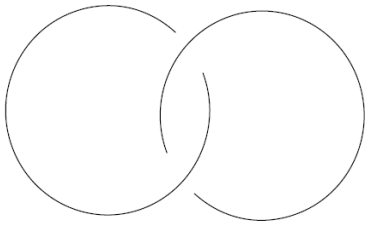


Figure 9. The Hopf Link.

The Borromean Link seen in Figure 10 is produced by linking three separate trivial links. The Borromean Link has the property that the three trivial links are linked in such a way that if one of those knots were to be removed, the remaining two trivial links would no longer remain linked together.

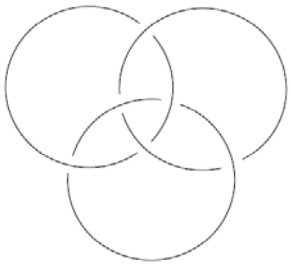


Figure 10. The Borromean Link.

The Whitehead Link has two common visual displays as noted in Figure 11. Please note that the trivial knot segments are labeled using t and the figure other knot's segments are labeled using e.

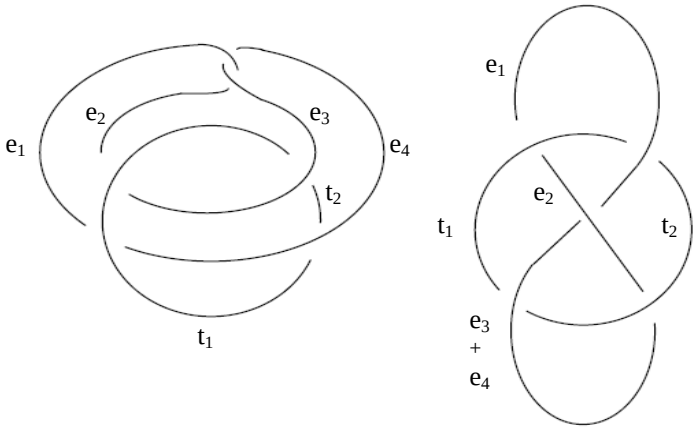


Figure 11. The Whitehead Link.

Two links are considered to be equivalent if and only if their images can be related by a finite sequence of Reidemeister moves [1]. Below we will prove that the two images of the Whitehead Link are in fact the equivalent.

We will begin with the image of the Whitehead Link found on the left side of Figure 11. We will start by rotating the trivial knot counterclockwise such that the crossing of the second knot is in the center of the trivial knot. Note that the number of segments for each knot is preserved. (Figure 12)

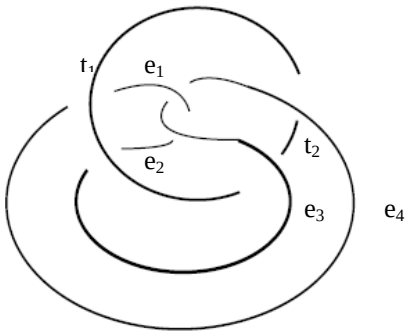


Figure 12. The Whitehead link after the first series of Reidemeister moves.

In the second step a type II transformation occurs causing the line segment e_4 be on the opposite side of the trivial link from the line segment e_3. This step incorporates both type II and III Reidemeister moves. Again, the number of segments for each knot is preserved. (Figure 13)

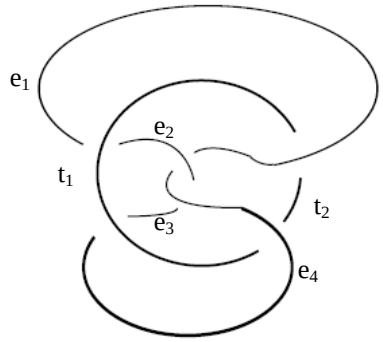


Figure 13. The Whitehead Link after the second series of Reidemeister moves.

Last, we perform a Type I Reidemeister move on the lower segments of the non-trivial knot. Doing so combines the segments e_3 and e_4 into the segment $e_3 + e_4$. All other segments are preserved. (Figure 14) Note that Figure 14 is the exact link as the right image in Figure 11. Thus, A finite sequence of Reidemeister moves has related the two links and the two links are equivalent.

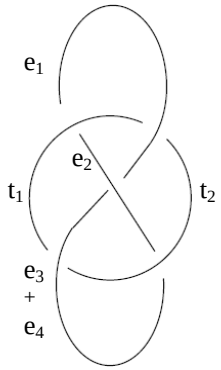


Figure 14. The Whitehead Link.

V. Conclusion

The triangle moves of the short cut and detour form the basis for the manipulation and transformation of knots. Kurt Reidemeister's further research in this area is an invaluable contribution to Knot Theory. This is because the Reidemeister moves replace all possible combinations of triangular moves that can be performed to simplify a knot. The Redemeister moves also allow for the simplification of both the definition for knot equivalence and link equivalence. Through two examples we have seen the application of Reidemeister moves to these definitions. Thus the Reidemeister moves make it easier to identify equivalent knots and equivalent links.

References

[1] Messer, Robert, and Phillip Straffin. *Topology Now!*. Washington, DC: The Mathematical Association of America, 2006.

[2] "Mathwords: Simple Closed Curve." 29 July, 2008. http://www.mathwords.com/s/simple_closed_curve.htm (accessed 12 March, 2009).

[3] Scriba, Cristoph J. "Reidemeister, Kurt Werner Friedrich." *Dictionary of Scientific Biography*. New York, Charles Scribner's Sons, 1975. p. 362-363.

Gregory A, George was born in the Hague, the Netherlands on April 6, 1987. He earned his Bachelor of Arts at Ripon College, Ripon, WI. His major field of study is mathematics.

He was commissioned as a Second Lieutenant in the U.S. Army on May 15, 2009. After completion of Infantry Basic Officer Leadership Course at Fort Benning, GA, his first duty station will be Alaska.