Compressive Sensing in Video Encoding

Brian Azizi

Department of Physics
Centre for Scientific Computing

University of Cambridge

This dissertation is submitted for the degree of

Master of Philosophy

Selwyn College July 2016

I would like to dedicate this thesis to my loving parents ...

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University. This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text. This dissertation contains less than 65,000 words including appendices, bibliography, footnotes, tables and equations and has less than 150 figures.

Brian Azizi
July 2016

Acknowledgements

And I would like to acknowledge ...

Abstract

This is where you write your abstract ...

Table of contents

List of figures

List of tables

Chapter 1

Introduction

There are three parts: A signal processing framework called *Compressive Sensing (CS)*, a pre-processing step in form of a basis transformation based on discrete wavelet transforms and a Machine Learning algorithm called *Sparse Bayesian Learning*.

Background

Chapter 2

Compressive Sensing

This section is based on [3]. The problem to be solved can be formulated as follows: Let $x \in \mathbb{R}^N$ be a signal of interest. We do not measure x directly and it is thus unknown. Instead, we have a measurement $y \in \mathbb{R}^M$, with $M << N$, from which we want to reconstruct x. The signals x and y are related as follows:

$$\Omega x = y \tag{2.1}$$

where Ω is a known $M \times N$ matrix referred to as the *sensing matrix*.

For example, in [3], the signal of interest x is an image, so that N is equal to the total number of pixels in the image and x_i is equal to the intensity of the corresponding pixel. However, we imagine that we have only access to a corrupted version of x in which random pixel values have been deleted. This is our measurement y. See Figure **??** for an example. The sensing matrix Ω corresponding to this scenario is obtained by taking the $N \times N$ identity matrix and deleting the rows that correspond to the missing entries in x.

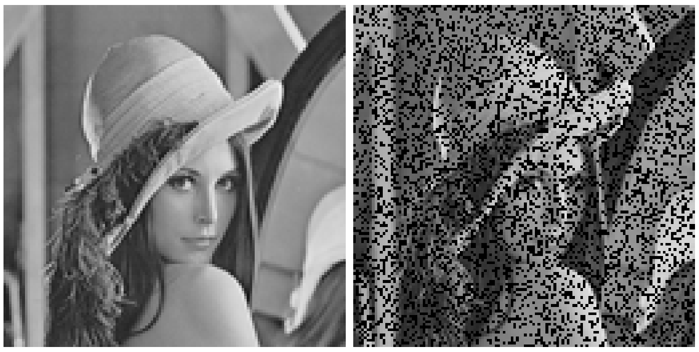

Fig. 2.1 Example of a signal pair x (left) and y (right). We wish to reconstruct x from y.

Compressive Sensing (CS) is a collection of signal processing techniques that allow for efficient *reconstruction* (and indeed *aquisition*) of such signals by solving the underdetermined system (2.1).

Of course, there are infinitely many solutions to an underdetermined system. In the CS framework, we seek to find a solution $\hat{x}$ that is *sparsest in some domain*. By that, we mean that we want to find $\hat{x}$ that satisfies (2.1), such that there exists a basis transformation of $\hat{x}$ in which it has the smallest number of nonzero entries.

More concretely, we assume there exists a domain in which the desired signal x is sparse. I.e. there exists a $N \times N$ basis matrix Ψ such that $x = \Psi w$ and w is sparse.

The CS problem can then be expressed as follows:

$$\min ||w||_0 \qquad \text{subject to} \qquad \Omega\Psi w = y \tag{2.2}$$

where $||.||$ denotes the l_0 norm, i.e. the number of nonzero components.

For a more detailed review of the CS framework, see [1].

Chapter 3

Wavelets

In this chapter, we will introduce wavelet functions and the *Discrete Wavelet Transform* (DWT). The DWT allows us to transform our signal into a new basis in which it is sparse.

$$x = \Psi w \tag{3.1}$$

that takes the dense signal x and sends it into a domain in which its transformation $w = \Psi^T x$ is sparse.

3.1 Discrete Cosine Transform

An aside on the DCT. Used in old JPEG standard (ref). Formulae. Interpretations. Pictures. However, before we discuss wavelets, we will briefly introduce the *Discrete Cosine Transform* (DCT)

3.2 Wavelet transforms

3.2.1 Haar wavelets

Finding a set of basis functions Ψ that achieve such a transformation lies at the heart of many lossy compression techniques. For instance, in image processing the JPEG 2000 standard is a widely used lossy compression technique that relies on this principle. The original image x is transformed into w using the so-called *Discrete Cosine Transform*. The basis matrix Ψ is orthogonal, so x and w have the same l_2 norm. In the original signal x the length is spread across many of its coefficients. On the other hand, most of the length of w is concentrated in a few of its coefficients. A large fraction of the entries in w are very close to zero. By deleting

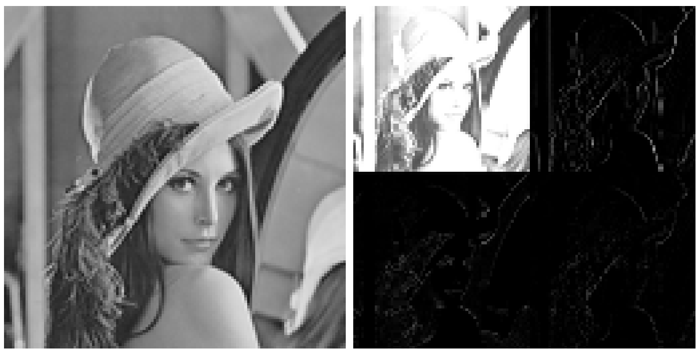
Fig. 3.1 Original image x (left) and its Haar basis transformation w (right). See next chapter for more details on Haar wavelets.

these entries in w and only storing the non-zero coefficients (and the corresponding basis functions), we can obtain a compressed version $\hat{w}$. This allows us to significantly reduce the amount of data that needs to be stored without affecting the visual quality in the reconstructed image $\hat{x} = \Psi\hat{w}$.

It is important to note here that the choice of basis functions Ψ typically has a significant effect on the performance of the reconstruction algorithms.

The simplest wavelet basis transformation is based on the Haar wavelets. Figure **??** vizualises a basis transformation of the original image x to w. This example uses a Haar wavelet basis at the first scale. We will explain the Haar wavelet transformations of images and videos in more detail in the next chapter. Dark areas correspond to small coefficients. Note that most entries in w are near zero. In practice, we approximate these entries as zero and treat w as sparse.

3.2.2 Daubechies Wavelets

Intuition. Where do the coeffs come from. Matrices. Boundary conditions.

3.3 Forming the basis matrix

1D, 2D, 3D case. Different scales

For a deeper introduction into wavelets see [4]. For more information on wavelet compression techniques, see [2].

Chapter 4

Sparse Bayesian Learning

So far, we have not addressed the central question: How do we solve the compressive sensing problem (2.2)? Various deterministic approaches have been developed in recent years. See [3] for an overview.

In the MPhil project, we will employ a probabilistic technique based on Sparse Bayesian Learning. In particular, we will use the *Relevance Vector Machine (RVM)* [5, 6] to reconstruct w from the measurements y. Following that, we obtain a reconstructed version of the desired signal x by pre-multiplying w by Ψ to obtain the desired signal.

The RVM is a *regression* technique. We model the relationship between y and x by

$$y_i = w^T \psi(x_i) + \varepsilon_i \tag{4.1}$$

where $\psi(x_i)$ is the ith column of Ψ and the ε_i are independent noise variables drawn from a zero-mean Gaussian distribution $\mathcal{N}(o, \sigma^2)$.

The model uses the available measurements $\{(x_1, y_1), \ldots, (x_M, y_M)\}$ as training data to train the model in a Bayesian framework. The result of training the RVM is a posterior distribution for the unknown vector of coefficients w. A special feature of the RVM is that the posterior mean μ of w is often very sparse.

In order to reconstruct the image, we use the estimated posterior mean to "predict" what a pixel value y^* should be at a location x^* in which information was missing:

$$y^* = w^T \psi(x^*) \tag{4.2}$$

Apart from achieving sparse solutions, one further desirable feature of the RVM is that the model provides error bars for its predictions. This is used in [3] to construct a multi-scale cascade of RVM estimations and achieve significant performance boosts.

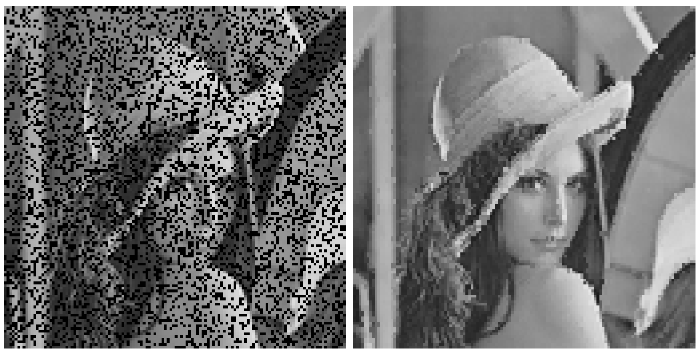

Fig. 4.1 Corrupted signal y (left) and reconstructed signal $\hat{x}$ (right) using a cascade of 3 RVMs with Haar basis functions (see [3]).

An example of this can be seen in Figure **??**.

For details on the RVM and its implementation see [3, 5, 6].

Chapter 5

Design of the Multi-Scale Cascade of Estimations Algorithm

Bringing all building blocks together. Description and explanation of the algorithm

5.1 Interpolator

We use a sensing matrix Ω that acts as signal mask. That is, we obtain the $N \times M$ matrix Ω by taking the $M \times M$ identity matrix I_M and deleting $(M - N)$ rows. This corresponds to a subsampled signal in which we only measured N pixel values. For this specific class of sensing matrices, the problem of reconstructing the original signal is also known as *interpolation*.

Chapter 6

Implementation Details and Code optimization

This chapter gives a brief description of the current state of our implementation of the 3D signal reconstructer.

6.1 Haar basis functions

The RVM takes as input a target vector (y) and a basis matrix (Ψ). In this respect, it is agnostic about whether the signal is an image or video or of some other type alltogether. Most of this information is encoded in the basis matrix Ψ. It is therefore important, and often challenging, to select a good set of basis functions.

Our current implementation uses 3-dimensional Haar wavelet basis functions. I will show how the basis matrix Ψ is constructed by briefly describing how the discrete Haar wavelet transform is performed on 1D, 2D and finally on 3D signals.

6.1.1 1D Haar wavelet transform

Consider a 1-dimensional signal $s = \{s_1, \dots, s_r\} \in \mathbb{R}^r$ (r for "rows"), where, for simplicity, we assume that r is a power of 2. The Haar wavelet transform can be performed at various resolution scales. The transform at the first scale is given by:

$$s = \{s_1, \dots, s_r\} \to \frac{1}{\sqrt{2}}\{s_1 + s_2, s_3 + s_4, \dots, s_{r-1} + s_r, s_1 - s_2, \dots, s_{r-1} - s_r\} = \hat{s}^{(1)}$$

The first half of the signal is replaced by scaled averages of adjacent elements and the second half is replaced by scaled differences of adjacent elements. By performing this transform again on the first half of $\hat{s}^{(1)}$ while keeping the second half fixed, we get the Haar wavelet transform at the second scale $\hat{s}^{(2)}$. To get the third scale transform $\hat{s}^{(3)}$, we perform the initial transform on the first quarter of $\hat{s}^{(2)}$ while keeping the rest of the signal fixed. We may continue this process until we reach the ith scale, where $2^i = r$.

From here on, we will only consider the first scale transform $\hat{s}^{(1)}$ and we will omit the (1) superscript. We can express the transform as a multiplication by an orthogonal $r \times r$ matrix W given by

$$W = \begin{bmatrix} \Phi_r \\ \Psi_r \end{bmatrix} \tag{6.1}$$

where Φ_r and Ψ_r are $(r/2) \times r$ matrices[1] given by

$$\Phi_r = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & 1 \end{pmatrix}$$

and

$$\Psi_r = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & -1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & -1 \end{pmatrix}$$

In the signal processing literature, Φ_r is referred to as a low pass filter, while Ψ_r is referred to as a high pass filter. Φ_r outputs an average of the signal and Ψ_r outputs the details of the signal.

6.1.2 2D Haar wavelet transform

Let $A \in \mathbb{R}^{r \times c}$ be a 2-dimensional signal (e.g. an image). For simplicity, we will assume that both r and c are powers of 2 (though not necessarily equal).

It is simple to obtain A's Haar wavelet transform $\hat{A}$ at the first scale. This is done by first applying the 1-dimensional transform individually to each column of A to obtain a temporary matrix $\hat{A}_{temp}$. Next, we apply the 1-dimensional haar wavelet transform individually to each row of $\hat{A}_{temp}$ to obtain $\hat{A}$.

[1]Note that the matrix Ψ_r used here is different to the matrix Ψ that was used in the previous chapter (which corresponds to W^T here).

We can again express the transform as a multiplication of matrices:

$$\hat{A} = \begin{bmatrix} \Phi_r \\ \Psi_r \end{bmatrix} A \begin{bmatrix} \Phi_c^T & \Psi_c^T \end{bmatrix} \tag{6.2}$$

where Φ_r and Ψ_r are as before and Φ_c and Ψ_c are of similar form but each have dimensions $(c/2) \times c$. This is the transform that was used to generate the RHS of Figure **??**. We note that the high-pass filters essentially detect edges of various orientations in the image.

However, as it currently stands, we cannot use this form of the basis transformation for the reconstruction algorithm. Recall that the RVM requires a *vector* of measurements as opposed to a matrix and also that it requires a single basis matrix, not a basis transform as given in (6.2).

To do this, we store the 2-dimensional signal A as a long column vector a of length rc by pasting the individual columns of A one after another. The basis transformation of a can then be expressed as

$$\hat{a} = Wa$$

where W is a $rc \times rc$ matrix given by

$$W = \begin{bmatrix} \Phi_c \otimes \Phi_r \\ \Phi_c \otimes \Psi_r \\ \Psi_c \otimes \Phi_r \\ \Psi_c \otimes \Psi_r \end{bmatrix}$$

The symbol $\otimes$ denotes the *Kronecker product*. The kronecker product $P \otimes Q$ between matrices P and Q with dimensions $m_P \times n_P$ and $m_Q \times n_Q$, respectively, is defined to be the block matrix

$$\begin{bmatrix} p_{1,1}Q & p_{1,2}Q & \cdots & p_{1,n_P}Q \\ p_{2,1}Q & p_{2,2}Q & \cdots & p_{2,n_P}Q \\ \vdots & \vdots & \ddots & \vdots \\ p_{m_P,1}Q & p_{m_P,2}Q & \cdots & p_{m_P,n_P}Q \end{bmatrix}$$

of size $m_P m_Q \times n_P n_Q$.

6.1.3 3D Haar wavelet transform

Let $V \in \mathbb{R}^{r \times c \times s}$ be a 3-dimensional signal such as a video. V has r rows, c columns and s slices, and we assume that r, c and s are all powers of 2. We may visualize V as a "volume" with 2 spacial dimensions and one time dimension corresponding to frames of the video.

To obtain the Haar wavelet transform $\hat{V}$ of V, we first perform the 1-dimensional transform individually on each column in every slice of V to get $\hat{V}_{temp1}$. We then perform the 1D transform on every row in every slice of $\hat{V}_{temp_1}$ to get $\hat{V}_{temp_2}$. Finally, we perform the 1D transform across the slices for every row and column to get $\hat{V}$.

However, like in the 2-dimensional case, we need to be able to pass a single vector of coefficients and a single basis matrix to the RVM. To do this, we vectorize V as follows. First, we vectorize each individual slice of V as before in the 2D case. Then, we stack all these vectors on top each other to get one very long column vector v of length rcs. The Haar wavelet transform is given by

$$\hat{v} = Wv$$

where

$$W = \begin{bmatrix} \Phi_s \otimes \Phi_c \otimes \Phi_r \\ \Phi_s \otimes \Phi_c \otimes \Psi_r \\ \Phi_s \otimes \Psi_c \otimes \Phi_r \\ \Phi_s \otimes \Psi_c \otimes \Psi_r \\ \Psi_s \otimes \Phi_c \otimes \Phi_r \\ \Psi_s \otimes \Phi_c \otimes \Psi_r \\ \Psi_s \otimes \Psi_c \otimes \Phi_r \\ \Psi_s \otimes \Psi_c \otimes \Psi_r \end{bmatrix}$$

Comparing notation to the previous chapter, what we refer to as W here is the transpose of what was previously denoted as Ψ. And since $v = W^T\hat{v}$, we see that v corresponds to what was previously called x.

Chapter 7

Results

We have obtained some results with our current implementation. The implementation uses the Haar wavelet transform at the first scale.

Our example video has a resolution of 128 by 128 pixels and consists of a total of 64 frames. Thus, $r = 128$, $c = 128$ and $s = 64$. Note that even for such a relatively small sample, the size of the basis matrix Ψ is $(128 * 128 * 64) \times (128 * 128 * 64) = 1048576 \times 1048576$. Even in single precision, storing this matrix would require around 4 terrabytes.

For this reason, we have split the original input signal into $8 \times 8 \times 8$ blocks and perform the algorithm on the individual blocks.

In Figures 3.1 and 3.2, we have included a sample frame from the corrupted test video and the same frame after reconstruction.

In Figure 3.1, we corrupted the video by deleting 30% of the pixel values in the first frame and deleting the same pixel values in each subsequent frame (so the same pixels are missing in each frame). Figure 3.2 uses the same corruption scheme but we deleted 50% rather than 30% of pixel values.

These initial results are promising, though clearly there are still improvements to be made.

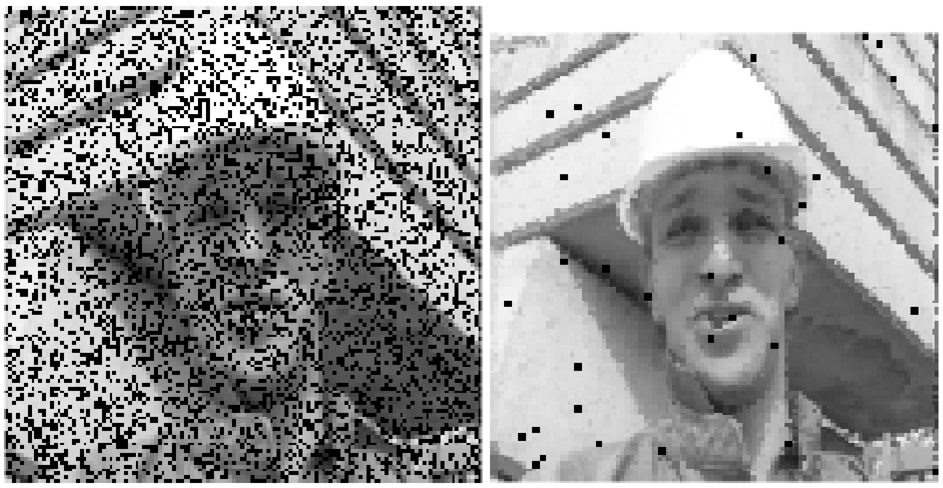

Fig. 7.1 Sample frame from corrupted video (left) and the reconstructed video (right)

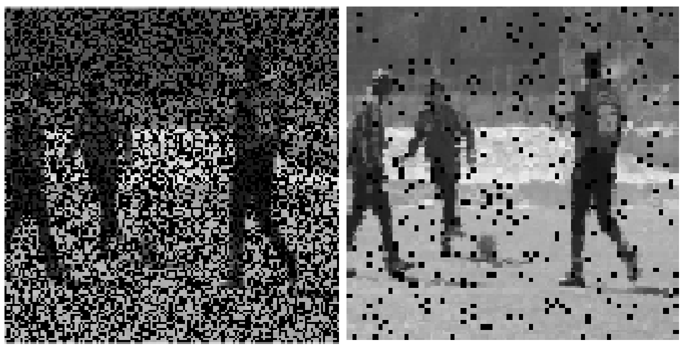

Fig. 7.2 Sample frame from corrupted video (left) and the reconstructed video (right)

Chapter 8

Conclusion

References

[1] Candès, E. J. and Wakin, M. B. (2008). An introduction to compressive sampling. *Signal Processing Magazine, IEEE*, 25(2):21–30.

[2] DeVore, R. A., Jawerth, B., and Lucier, B. J. (1992). Image compression through wavelet transform coding. *Information Theory, IEEE Transactions on*, 38(2):719–746.

[3] Pilikos, G. (2014). Signal reconstruction using compressive sensing. MPhil thesis, University of Cambridge.

[4] Stollnitz, E. J., DeRose, T. D., and Salesin, D. H. (1995). Wavelets for computer graphics: a primer. 1. *Computer Graphics and Applications, IEEE*, 15(3):76–84.

[5] Tipping, M. E. (2001). Sparse bayesian learning and the relevance vector machine. *The journal of machine learning research*, 1:211–244.

[6] Tipping, M. E., Faul, A. C., et al. (2003). Fast marginal likelihood maximisation for sparse bayesian models. In *AISTATS*.

Appendix A

How to install LaTeX

Windows OS

TeXLive package - full version

1. Download the TeXLive ISO (2.2GB) from
   https://www.tug.org/texlive/
2. Download WinCDEmu (if you don't have a virtual drive) from
   http://wincdemu.sysprogs.org/download/
3. To install Windows CD Emulator follow the instructions at
   http://wincdemu.sysprogs.org/tutorials/install/
4. Right click the iso and mount it using the WinCDEmu as shown in
   http://wincdemu.sysprogs.org/tutorials/mount/
5. Open your virtual drive and run setup.pl

or

Basic MikTeX - TeX distribution

1. Download Basic-MiKTeX(32bit or 64bit) from
   http://miktex.org/download
2. Run the installer
3. To add a new package go to Start » All Programs » MikTex » Maintenance (Admin) and choose Package Manager

4. Select or search for packages to install

TexStudio - Tex Editor

1. Download TexStudio from
   http://texstudio.sourceforge.net/#downloads

2. Run the installer

Mac OS X

MacTeX - TeX distribution

1. Download the file from
   https://www.tug.org/mactex/

2. Extract and double click to run the installer. It does the entire configuration, sit back and relax.

TexStudio - Tex Editor

1. Download TexStudio from
   http://texstudio.sourceforge.net/#downloads

2. Extract and Start

Unix/Linux

TeXLive - TeX distribution

Getting the distribution:

1. TexLive can be downloaded from
   http://www.tug.org/texlive/acquire-netinstall.html.

2. TexLive is provided by most operating system you can use (rpm,apt-get or yum) to get TexLive distributions

Installation

1. Mount the ISO file in the mnt directory

   ```
   mount -t iso9660 -o ro,loop,noauto /your/texlive####.iso /mnt
   ```

2. Install wget on your OS (use rpm, apt-get or yum install)

3. Run the installer script install-tl.

   ```
   cd /your/download/directory
   ./install-tl
   ```

4. Enter command ‘i’ for installation

5. Post-Installation configuration:
   http://www.tug.org/texlive/doc/texlive-en/texlive-en.html#x1-320003.4.1

6. Set the path for the directory of TexLive binaries in your .bashrc file

For 32Bit OS

For Bourne-compatible shells such as bash, and using Intel x86 GNU/Linux and a default directory setup as an example, the file to edit might be

```
edit $~/.bashrc file and add following lines
PATH=/usr/local/texlive/2011/bin/i386-linux:$PATH;
export PATH
MANPATH=/usr/local/texlive/2011/texmf/doc/man:$MANPATH;
export MANPATH
INFOPATH=/usr/local/texlive/2011/texmf/doc/info:$INFOPATH;
export INFOPATH
```

For 64Bit

```
edit $~/.bashrc file and add following lines
PATH=/usr/local/texlive/2011/bin/x86_64-linux:$PATH;
export PATH
MANPATH=/usr/local/texlive/2011/texmf/doc/man:$MANPATH;
export MANPATH
```

```
INFOPATH=/usr/local/texlive/2011/texmf/doc/info:$INFOPATH;
export INFOPATH
```

Fedora/RedHat/CENTOS:

```
sudo yum install texlive
sudo yum install psutils
```

SUSE:

```
sudo zypper install texlive
```

Debian/Ubuntu:

```
sudo apt-get install texlive texlive-latex-extra
sudo apt-get install psutils
```

Appendix B

Installing the CUED Class file

LATEX.cls files can be accessed system-wide when they are placed in the <texmf>/tex/latex directory, where <texmf> is the root directory of the user's TEXinstallation. On systems that have a local texmf tree (<texmflocal>), which may be named "texmf-local" or "localtexmf", it may be advisable to install packages in <texmflocal>, rather than <texmf> as the contents of the former, unlike that of the latter, are preserved after the LATEXsystem is reinstalled and/or upgraded.

It is recommended that the user create a subdirectory <texmf>/tex/latex/CUED for all CUED related LATEXclass and package files. On some LATEXsystems, the directory look-up tables will need to be refreshed after making additions or deletions to the system files. For TEXLive systems this is accomplished via executing "texhash" as root. MIKTEXusers can run "initexmf -u" to accomplish the same thing.

Users not willing or able to install the files system-wide can install them in their personal directories, but will then have to provide the path (full or relative) in addition to the filename when referring to them in LATEX.